VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Knihovna evolučních algoritmů v Javě
Evolution Algorithms Library in Java

2015 Petr Štěpánek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Petr Štěpánek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Knihovna evolučních algoritmů v Javě
Evolution Algorithms Library in Java

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit funkční knihovnu pro evoluční algoritmy v programovacím jazyce Java.

1. Student nastuduje odbornou literaturu z oblasti evolučních algoritmů.
2. Student naimplementuje tyto evoluční algoritmy:
   - Particle Swarm
   - Diferenciální evoluce
   - SOMA
   - Simulované žíhání
   - Evoluční strategie
   a další
3. Student naimplementuje prototyp, kde demonstruje funkčnost vytvořené knihovny - prototyp bude obsahovat grafické uživatelské rozhraní, kde si uživatel bude moci zvolit algoritmus a jeho parametry.
4. Student vytvoří přehlednou dokumentaci.

Seznam doporučené odborné literatury:

[1] Oplatková, Zuzana; Ošmera, Pavel; Šeda, Miloš; Včelař, František; Zelinka, Ivan: Evoluční výpočetní techniky - principy a aplikace, BEN-Technická literatura, 80-7300-218-3, 2008
[2] Zelinka, Ivan: Umělá inteligence - hrozba nebo naděje?, BEN-Technická literatura,80-7300-068-7, 2003
[3] Zelinka, Ivan: Umělá inteligence v problémech globální optimalizace, BEN - Technická literatura, 80-7300-069-5, 2002
[4] Kenneth Price, Rainer M. Storn, Jouni A. Lampinen: Differential evolution a practical approach to global optimization, Natural Computing Series, 978-3-540-31306-9, 2005
[5] Kennedy, J.: Particle swarm optimization, Neural Networks, 1995. Proceedings., IEEE International Conference on, 0-7803-2768-3, 1995

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Lenka Skanderová**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne: 4.5.2015 ..... Štěpánek .....

Podpis

Poděkování

Rád bych poděkoval slečně Ing. Lence Skanderové za odbornou pomoc, doporučení a konzultace při vytváření této práce.

Panu Ing. Michalovi Kravčenkovi za návrh testovací funkce, která se odlišuje od ostatních doporučených testovacích funkcí a další odbornou pomoc.

Abstrakt

Cílem práce je vytvořit knihovnu evolučních algoritmů v jazyce Java, kterou je možno snadno importovat a použít v projektech. V knihovně bude také možnost nastavovat parametry pro pokročilé ovládání. Pro demonstraci knihovny bude také vytvořen jednoduchý program s grafickým rozhraním, ve kterém bude možno ukázat základní funkce knihovny.

Klíčová slova
Java, Evoluční algoritmy, Testovací funkce

Abstract

The aim of this work is to create java library of evolution algorithm, which can be easily imported and used in projects. There will be possibility to set parameters in library for advanced usage. The simple program with graphical interface will be made for demonstration of library, which will be able to perform basic functions of java library.

Key words

Java, Evolution algorithms, Test functions

Seznam použitých symbolů a zkratek

Java	Multiplatformní objektově orientovaný programovací jazyk
Evoluční algoritmy	Algoritmy, které se snaží využít modelů evolučních procesů, aby tak nalezly řešení náročných a rozsáhlých úloh v co nejkratším čase
Testovací funkce	Funkce vyjádřené rovnicí, jejichž graf je n dimenzionální hyperplocha, na které se testuje funkčnost a výkonnost evolučních algoritmů
Jedinec	Nebo také anglicky Specimen - je instance, se kterou pracují evoluční algoritmy pro nalezení optimálního řešení. Jedinec má chromozomy, které jsou v průběhu evoluce měněny
Populace	Sada jedinců, kteří spolu vzájemně operují, pro nalezení globálního extrému ve funkci
Vhodnost	Vychází z anglického fitness a označuje kvalitu jedince. V této knihovně se evoluční algoritmy snaží najít co nejmenší číslo s vhodností jedince, což znamená, že je jedinec nejlepší z populace
Default	Anglické slovo, které se využívá v programování - je jím myšleno základní nastavení, tzn. že defaultní hodnota je základní hodnota čísla nastavená přímo v kódu
JDK	Java Development Kit neboli soubor nástrojů pro vývoj aplikací pro platformu Java
PSO	Zkratka využívající se pro označení algoritmu rojení částic. Vychází z anglického Particle Swarm Optimalization
DE	Zkratka označující diferenciální evoluce
SOMA	Zkratka označující optimalizační algoritmus SamoOrganizujicí se Migrační Algoritmus

Obsah

1. Úvod

1.1. O práci

Práce se zabývá vytvořením knihovny evolučních algoritmů v Java, která bude snadno použitelná v jiných projektech a hlavně jednoduše rozšiřitelná o další evoluční algoritmy a prohledávané funkce. Práce obsahuje různé variace evolučních algoritmů typu rojení částic, diferenciálních evolucí, genetických algoritmů a simulovaného žíhání. Dále je v práci několik testovacích funkcí, primárně určených pro rychlé odzkoušení nových evolučních algoritmů.

Existuje několik dalších projektů evolučních algoritmů v Java, jako například Watchmaker Framework [9], Jenes [10], nebo Ejc [11]. Každá knihovna je využitelná k něčemu jinému, Watchmaker Framework [9] je pouze Framework, který umožňuje snadněji vyrobit vlastní evoluční algoritmus, jelikož na vysoké úrovni řeší spoustu základních problémů jako multi-threading, náhodný výběr, výběr nejlepších a další potřebné funkce. Jenes [10] se zase specializuje na genetické algoritmy. Ejc [11] je již rozsáhlá knihovna algoritmů, obsahuje diferenciální evoluce, evoluční strategie, PSO či algoritmus optimalizace mravenčí kolonií. Na druhou stranu neobsahuje mnoho variací těchto algoritmů. Například diferenciální evoluce najdeme v knihovně Jenes [10] pouze dvě. Pro srovnání v této práci jich je šestnáct. Žádná ze jmenovaných knihoven neobsahuje algoritmy SOMA, ani předpřipravenou testovací funkci s nedefinovanými oblastmi uvnitř prohledávaných hranic.

Práce se snaží klást důraz na jednoduché zapojení knihovny do projektu a snaží se minimalizovat operace, které uživatel musí provést předtím, než spustí evoluční algoritmus na vlastní prohledávané funkci. Knihovnu je také možno snadno rozšiřovat o další evoluční algoritmy.

Knihovna navíc obsahuje algoritmy, které nejsou v jiných knihovnách takto pohromadě, jako například algoritmy SOMA a exponenciální i binomickou verzi genetických algoritmů, nebo také nové verze diferenciálních evolucí.

Jelikož bylo potřeba testovat optimalizační funkce při jejich vývoji, vzniklo několik testovacích funkcí jako součást knihovny. Nakonec byly naimplementovány 4 testovací funkce, které byly vybrány pro jejich vzájemnou velkou odlišnost a velmi specifické až extrémní vlastnosti a dalších 11 multimodálních testovacích funkcí pro obecnější testování evolucí mimo extrémní případy. Funkce byly vybrány na základě benchmarku CEC2014 [1], jelikož v něm jsou používány.

Pro implementaci byla použita Java JDK 7. Zdrojové kódy jsou napsány, kompilovány a distribuovány v nástroji NetBeans 8.0.2.

1.2. O Evolučních algoritmech

Evoluční algoritmy jsou založeny na Darwinově a Mendelově evoluční teorii, spojující myšlenku předávání rodičovského genu svým potomkům, evoluce druhu a přirozeného výběru. Tedy stejně jako v přírodě přežijí a množí se jen ti jedinci, kteří mají správné vlastnosti - například jsou silnější, větší a chytřejší; v evolučních algoritmech generují potomky jen ti jedinci z populace, kteří mají nejlepší vlastnosti, tzn. souřadnice, které v dané funkci dají ty nejlepší výsledky. Tito potomci, stejně jako v přírodě, mutují a je-li tato evoluce posun k lepšímu, mají větší šanci přežít jak v přírodě, tak v evolučních algoritmech.

1.3. Vylepšení algoritmů v průběhu času

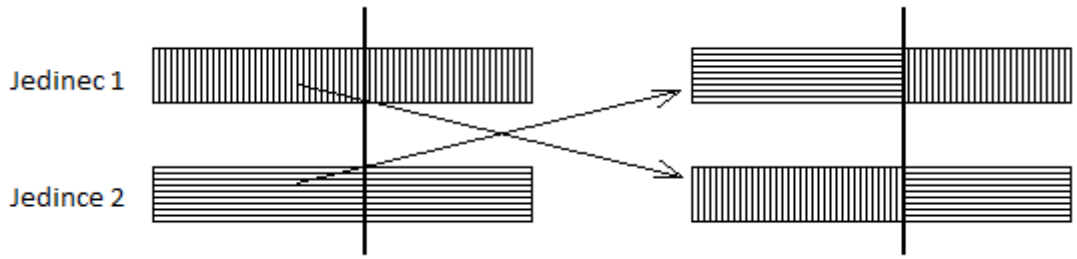
Přestože je evolučním algoritmům věnována pozornost až v posledních letech, počátky evolučních algoritmů se datují od 60 let dvacátého století, kdy se nezávisle na sobě objevilo asi deset projektů např. práce Bremermanna [13], která se zabývá otázkou, jak zvýšit produktivitu pomocí počítačů, Friedberga [14], která se zabývá problematikou sebe učením se a Boxe [15], která se zabývá evolucí inspirovanou přírodou. Rostoucí pozornost je zapříčiněna zvyšujícím se výpočetním výkonem [12].

V práci se věnuji především algoritmu diferenciální evoluce a jeho variacím, evolučním strategiím, genetickým algoritmům a rojení částic.

1.3.1. Genetické algoritmy

Poprvé byly představeny v roce 1975 Johnem Hollandem [17]. Jsou založeny na pozorování evoluce živých organismů. Odtud se převzala terminologie jako jedinec, generace, geny.

V knihovně je naimplementovaná verze s jednobodovým křížením, hybridní verze, která vždy po několika generacích provede lokální optimalizaci pomocí horolezeckého algoritmu a paralelní verze, dělící jedince do několika populací, které si mezi sebou vyměňují dva náhodně vybrané jedince.



Obrázek 1 Jednobodové křížení

1.3.2. Algoritmus diferenciální evoluce a jeho variace

Diferenciální evoluce byla představena v roce 1995 Kenem Pricem a Rainerem Stornem [21] a vycházela z algoritmu genetického žíhání, který byl publikován Stormem v magazínu Dr. Dobb's. Tento algoritmus prošel změnami jako převedení z binární do dekadické soustavy, přidání zkušebního vektoru a mutace. Také byly vypuštěny principy žíhání, jelikož byli nadbytečné. Postupem času se rozšířily, modifikovaly a dnes jsou jedny s nejsilnějších optimalizačních algoritmů.

Tyto evoluce se právě díky své síle často využívají v praxi, například v práci „*Reconstruction of two-dimensional buried objects by a differential evolution method*" [42] nebo pro návrh IIR filtru [43].

V knihovně je poměrně velké množství variací diferenciálních evolucí od starších - standardních verzí, po novější verze jako JADE [18], nebo SADE [19,20]. Konkrétními rovnicemi a vlastnostmi pro jednotlivé variace se zabývá část 4.3.

1.3.3. Evoluční strategie

Evoluční strategie byly vyvinuty v roce 1965 výzkumníky Bienertem, Rechenbergem a Schwefelem, se zaměřením na problémy z oblasti strojního inženýrství v Berlíně [22]. Variace těchto strategií se postupem času objevilo několik. Dvoučlenné, vícečlenné, rekombinační a adaptační [23]. Jejich principy se zabývá kapitola 4.4

1.3.4. Rojení částic

Optimalizace rojením částic také pracuje s jedinci, ale je inspirována sociálním chováním živočichů žijících v hejnech jako ptáci, hmyz či ryby. Rojení částic bylo vyvinuto Eberhartem a Kennedym v roce 1995 [24]. Jedná se tedy o nejmladší techniku optimalizace v této knihovně. Jednotlivé variace roje částic jsou rozepsány v kapitole 4.6

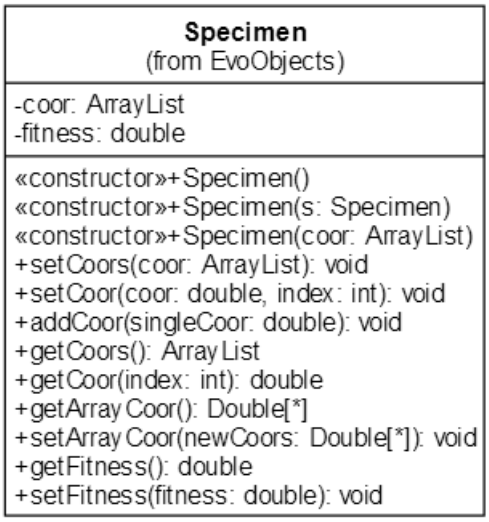
Speciálním případem optimalizace rojením částic jsou algoritmy SOMA, které vznikly roku 1999. Při jejich migracích se netvoří noví jedinci stejně jako u roje částic a jeho činnost se zakládá na vektorových operacích. S populací pracují podobně jako genetické algoritmy. Myšlenka je založena na skupině jedinců kooperujících při řešení optimalizačního problému - inspirováno organismy pracující ve větších skupinách, jako jsou včely, mravenci či predátoři [2,25]. Jednotlivé variace použité v knihovně jsou popsány v kapitole 4.7.

2. Vlastní třídy

2.1. Specimen

Instance této třídy představuje jednoho jedince v populaci. Celá populace jedinců je předávána evolučním algoritmům, které pomocí populace hledají extrémy funkce. Je to základní třída, která se předává do všech evolučních funkcí.

2.1.1. Reprezentace třídy

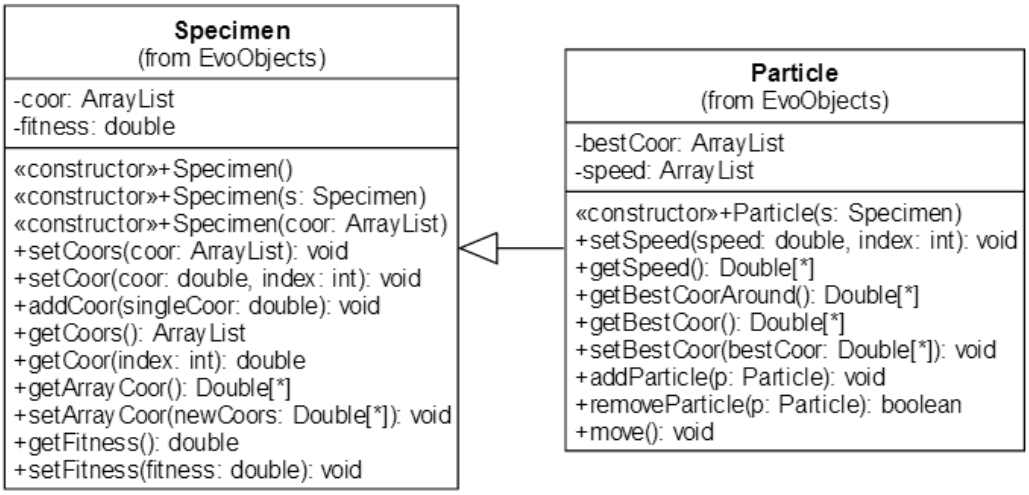


Obrázek 2 Reprezentace třídy – Specimen

2.2. Particle

Všechny implementované typy optimalizačního algoritmu rojení částic potřebují pro svou práci jedince, kteří mají více vlastností, než obsahuje třída Specimen. Proto vznikla třída Particle jakožto potomek třídy Specimen. Tuto třídu není pro práci s PSO nutno definovat uživatelem. Třída se vytváří na základě třídy Specimen přímo v PSO algoritmech před evolucí. Po evoluci se třída převede zpět na třídu Specimen, jejichž seznam se na konci evoluce vrací.

2.2.1. Třídní diagram



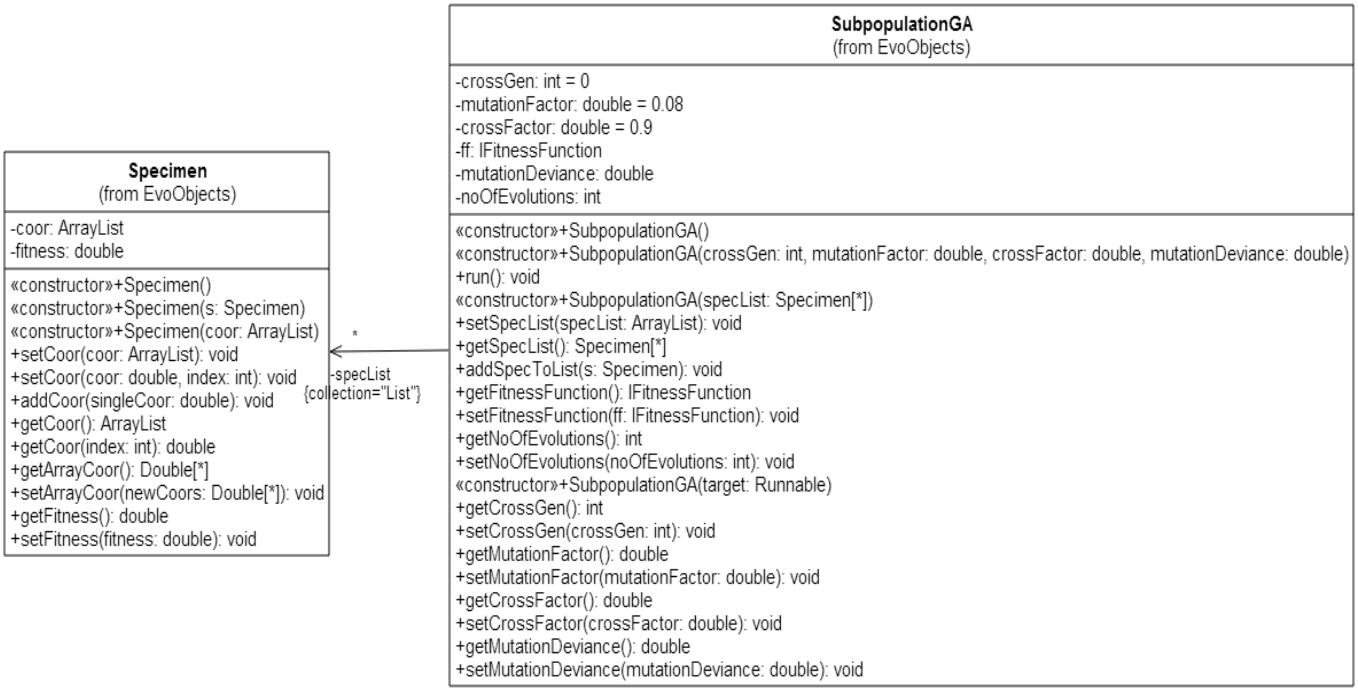
Obrázek 3 Třídní diagram – Particle

2.3. SubpopulationGA

Paralelní genetický algoritmus pro svou práci potřebuje rozdělit populaci jedinců na několik podpopulací. Za tímto účelem vznikla tato třída.

Za zmínku stojí parametr mutationDeviance určující maximální odchylku při mutaci. Pokud je nastavená v záporných hodnotách, je odchylka nastavena automaticky jako desetina velikosti prohledávané oblasti. V opačném případě je nastavena uživatelem dle jeho potřeb.

2.3.1. Třídní diagram

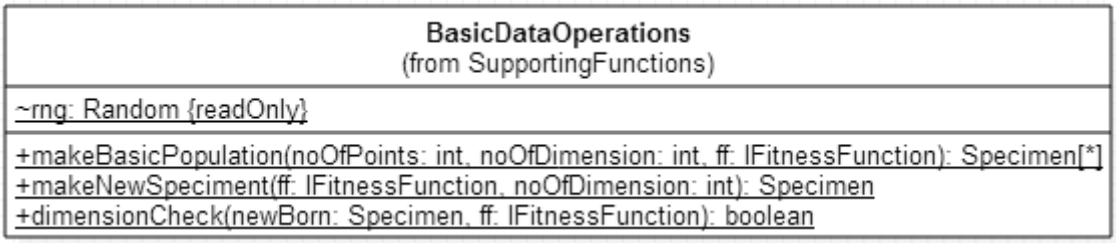


Obrázek 4 Třídní diagram - SubpopulationGA

2.4. BasicDataOperations

Tato třída se stará o základní úlohy před a během evoluce. Konkrétně vytváří nové jedince v prohledávaném prostoru, vytváří celou populaci jedinců pro původní inicializaci a kontroluje, zda je daný jedinec v rozmezí prohledávané oblasti.

2.4.1. Reprezentace třídy



Obrázek 5 Reprezentace třídy - BasicDataOperations

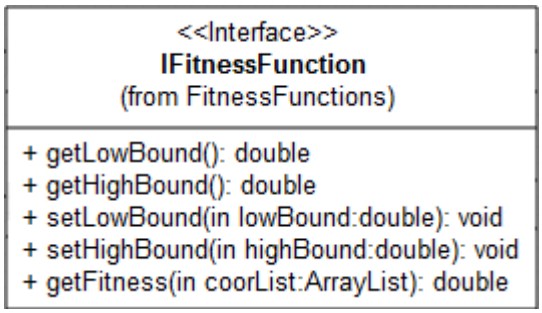
3. Testovací funkce

V knihovně jsou také implementované testovací funkce, primárně určené pro ověření funkčnosti a výkonu evolučních algoritmů během jejich implementace.

3.1. Rozhraní IFitnessFunction

Všechny testovací funkce jsou v tomto rozhraní, aby se zajistilo jednoduché nastavení funkcí v demonstrační desktopové aplikaci. Implementované optimalizační funkce mají toto rozhraní jako svůj parametr, proto je vhodné nové testovací funkce psát pod tímto rozhraním.

3.1.1. Reprezentace rozhraní

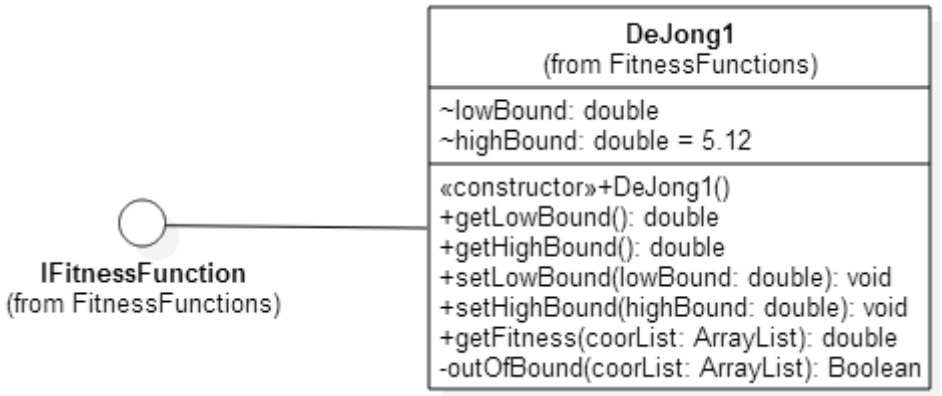


Obrázek 6 Reprezentace rozhraní – IFitnessFunction

3.2. DeJong1

Tvarem a předpisem testovací funkce DeJong1 se zabývá práce „*Test functions for optimization needs*“ [27]. Do knihovny byl zvolen pro jeho jednoduchost a jasnou viditelnost, zda algoritmus postupuje k minimu správným směrem.

3.2.1. Třídní diagram

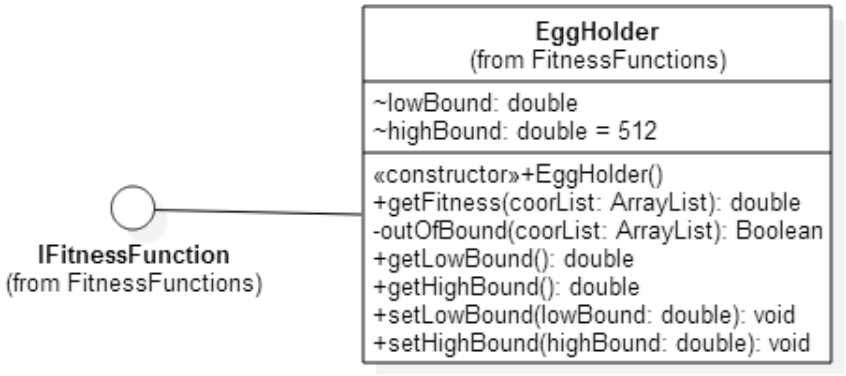


Obrázek 7 Třídní diagram - DeJong1

3.3. EggHolder

Testovací funkcí EggHolder, jejím tvarem a předpisem se zabývá práce „*Some New Test Functions for Global Optimization and Performance of Repulsive Particle Swarm Method*“ [26]. Do knihovny byl zařazen pro velké množství lokálních extrémů, protože se jeho globální extrém mění s velikostí prohledávané oblasti, a také proto, že jeho globální extrém není v nule.

3.3.1. Třídní diagram



Obrázek 8 Třídní diagram –EggHolder

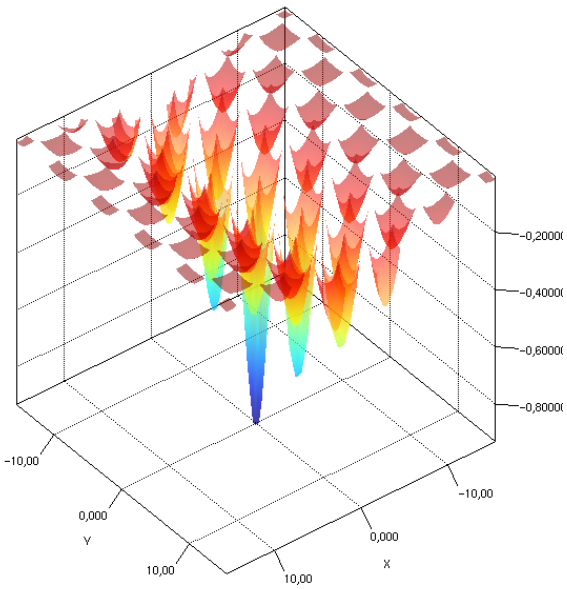
3.4. KravcenkoM

Tuto funkci matematicky napsal Ing. Michal Kravčenko na mou žádost, jelikož jsem hledal funkci, která bude mít v prohledávaném prostoru nedefinované území, aby bylo možno otestovat, zda prohledávací algoritmy správně pracují i v případě, kdy pro danou polohu jedince není z matematického pohledu definována jejich vhodnost.

Matematický předpis této funkce: $A = \prod_{i=1}^{D} \cos x_i \,;\; B = \sum_{i=1}^{D} \frac{x_i^2}{15^i} \,;\; f(x) = \frac{\sqrt[D]{A}}{e^{\sqrt{B}}}$

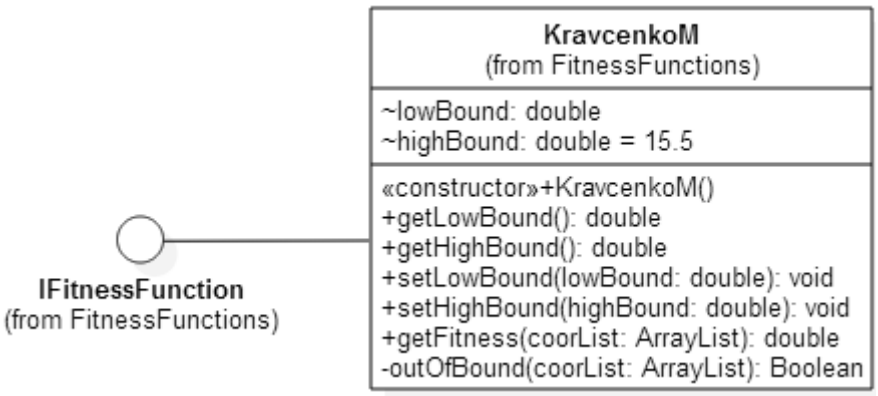
Přednastavené hranice prohledávané oblasti jsou $\langle -15.5; 15.5 \rangle$.

3.4.1. Graf funkce



Obrázek 9 3D graf pro 2D funkci - KravcenkoM

3.4.2. Třídní diagram



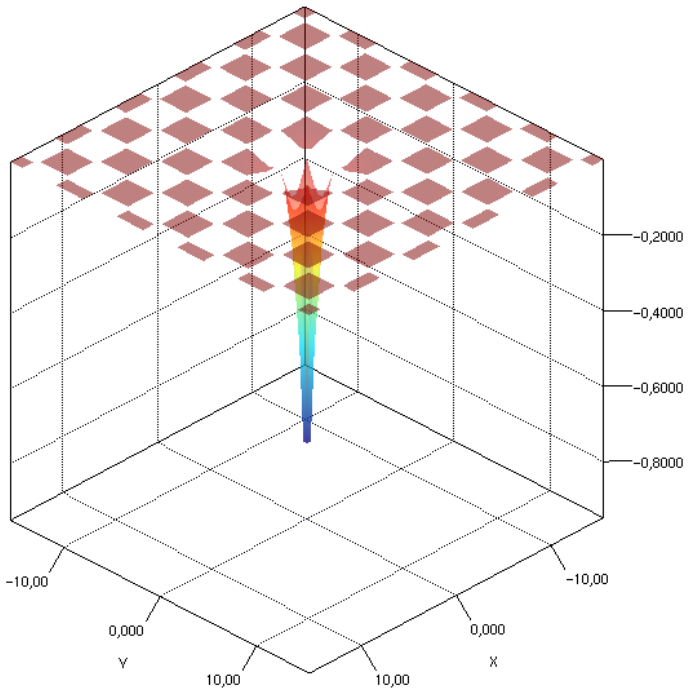
Obrázek 10 Třídní diagram - KravcenkoM

3.5. KravcenkoM2

Tato třída je totožná s třídou KravcenkoM z kapitoly 3.4. Pouze její matematický předpis byl pozměněn tak, že je funkce všude mimo střed téměř rovná nule, takže jedinci v této oblasti nemají žádné vodítko, kde je globální extrém.

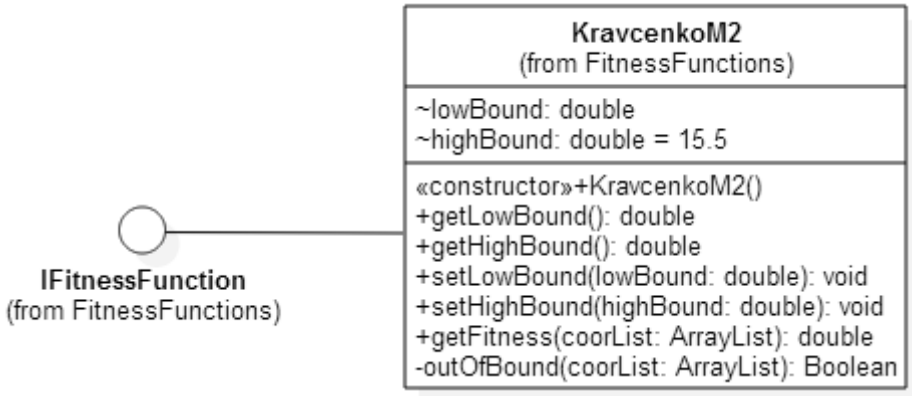
Matematický předpis této funkce: $A = \prod_{i=1}^{D} \cos x_i \; ; \; B = \sum_{i=1}^{D} x_i^2 \; ; \; f(x) = \frac{\sqrt[D]{A}}{e^{\sqrt{B}}}$

3.5.1. Graf funkce



Obrázek 11 3D graf pro 2D funkci - KravcenkoM2

3.5.2. Třídní diagram



Obrázek 12 Třídní diagram - KravcenkoM2

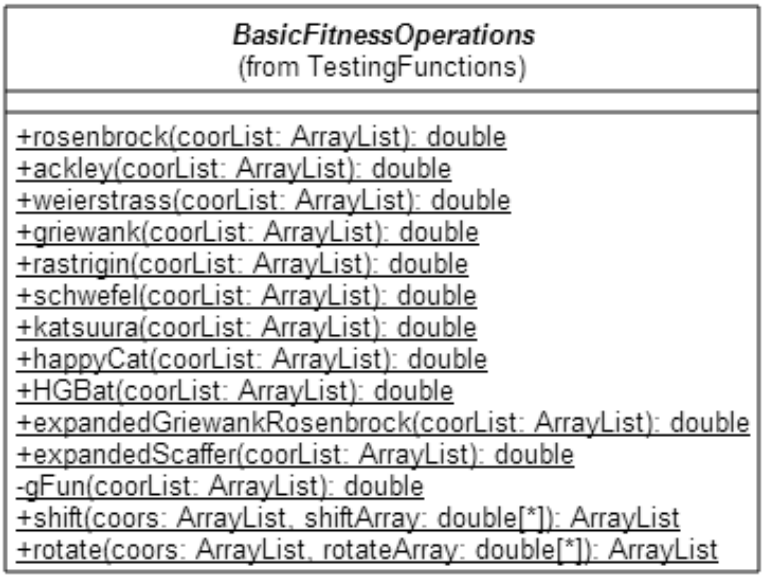
3.6. Multimodální funkce

Do projektu bylo zakomponováno dalších 11 multimodálních funkcí pro další potřeby testování algoritmů. Všechny tyto funkce nalezneme v abstraktní třídě BasicFitnessOperations. Každá tato funkce se volá ze speciální třídy, která je vhodná pro případné rozšíření knihovny o některý z benchmarků. Tyto funkce byly vybrány konkrétně na základě benchmarku CEC2014, jehož problematikou se zabývají J. J. Ling, B. Y. Qu, P. N. Suganthan [1]. Tento benchmark byl inspirací kvůli svojí náročnosti nalezení globálního extrému.

3.6.1. BasicFitnessOperations

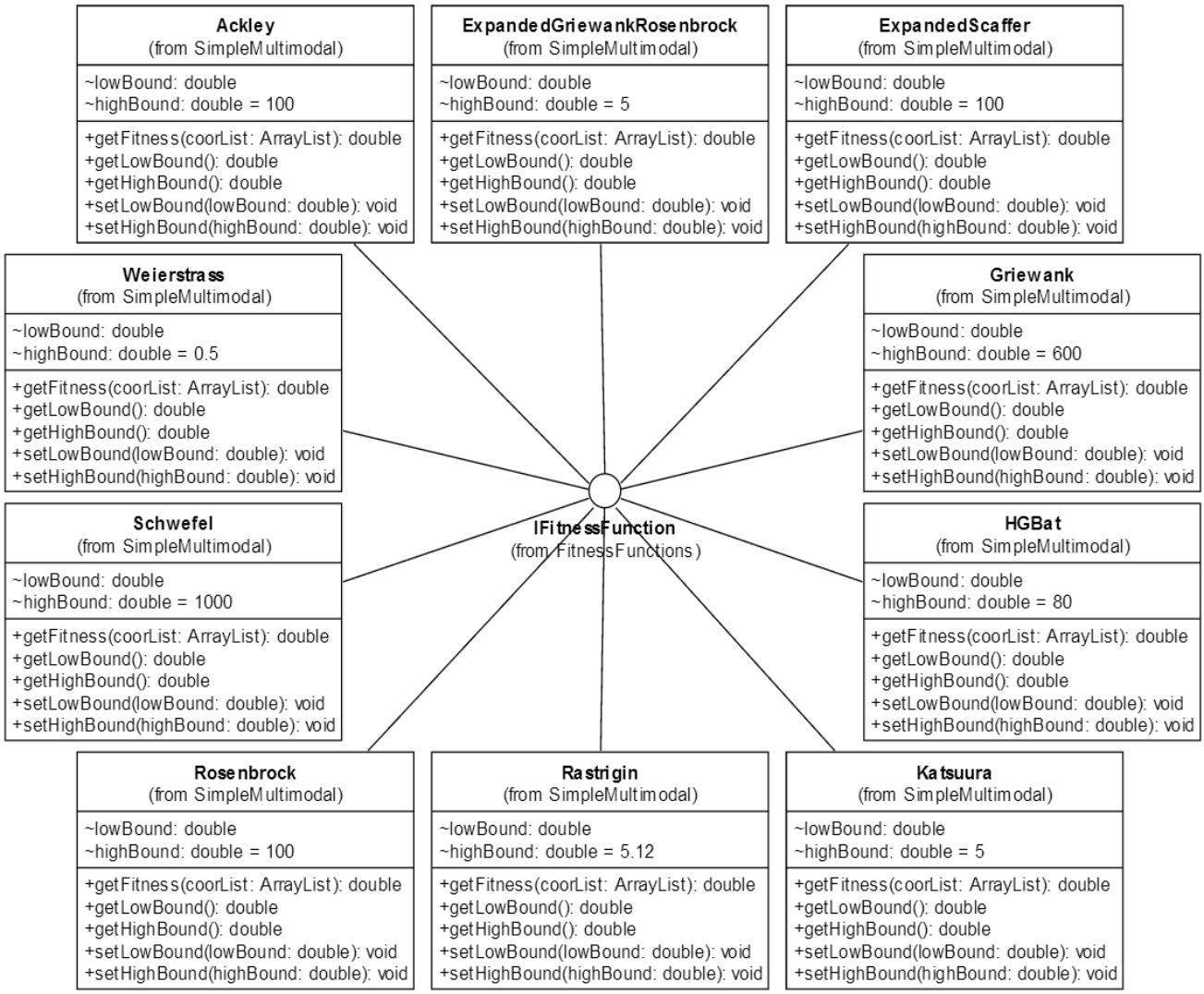
Tato třída obsahuje přepisy pro 11 multimodálních funkcí, předpřipravené funkce pro rotaci a posun, které se v projektu zatím nevyužívají. Názvy funkcí, které lze vidět na Obrázek 13 Reprezentace třídy - BasicFitnessOperations, odpovídají názvům implementovaných multimodálních funkcí.

3.6.1.1. Reprezentace třídy



Obrázek 13 Reprezentace třídy - BasicFitnessOperations

3.6.1.2. Třídní diagram multimodálních testovacích funkcí



Obrázek 14Třídní diagram - Multimodální funkce

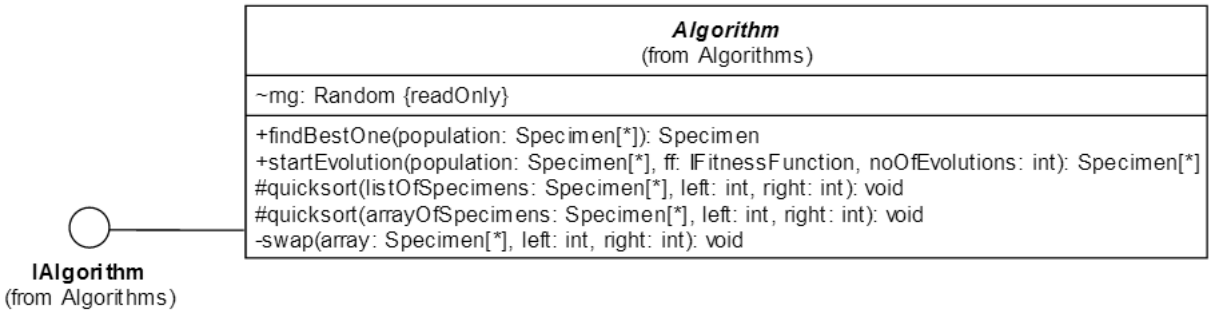
4. Algoritmy

Všechny evoluční algoritmy spadají pod interface IAlgorithm pro udržení standardu případného rozšíření knihovny a snadného importování do testovací grafické aplikace. Každý algoritmus v knihovně je vytvořený pro hledání globálního minima, proto je třeba dát si pozor při vytváření vlastní prohledávané funkce a uvědomit si, co chceme najít, případně celou funkci vynásobit minus jedničkou, aby se z globálního maxima stalo minimum a naopak. Všechny algoritmy jsou konstruované pro dvě a více dimenzí.

4.1. Třída Algorithm

Algorithm je abstraktní třída, která implementuje rozhraní IAlgorithm. Pro případné další rozšíření knihovny se doporučuje, aby nové algoritmy vznikaly jakožto potomci této třídy, jelikož má v sobě zabudovaný generátor náhodných čísel, QuickSort pro seřazení jedinců v poli od největší vhodnosti po nejmenší a řeší nalezení nejlepšího jedince v populaci. Funkci *startEvolution* pro zahájení prohledávání je potřeba v konkrétních algoritmech přepsat vlastním kódem.

4.1.1. Třídní diagram

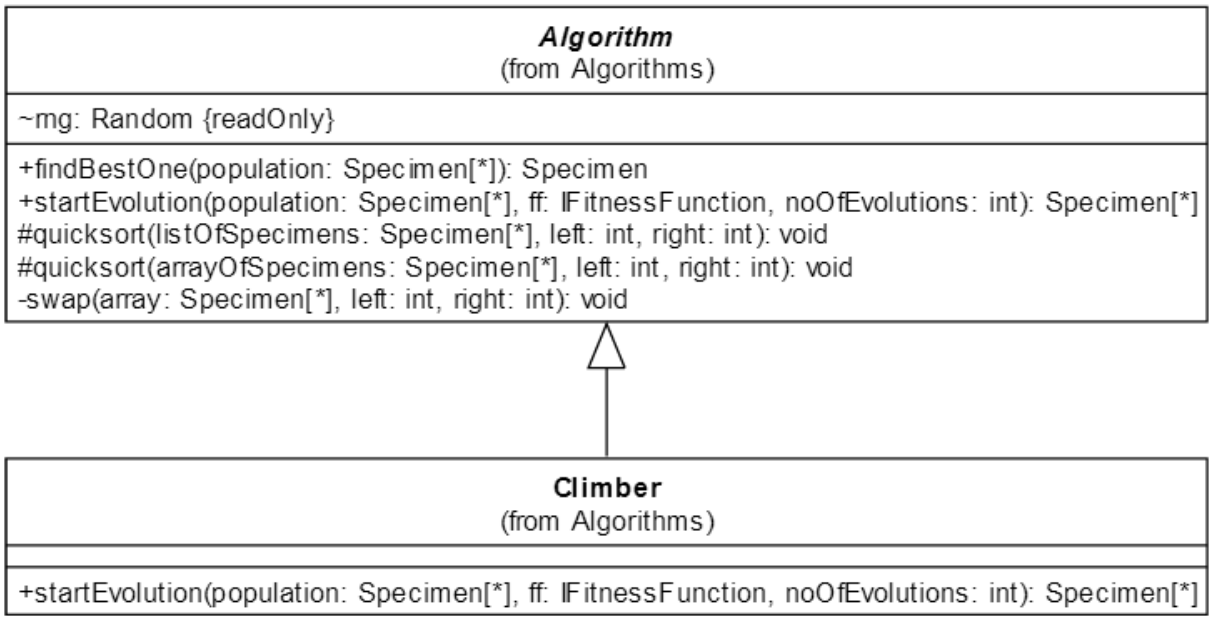


Obrázek 15 Třídní diagram - Algorithm

4.2. Horolezecký algoritmus

Evoluční algoritmus hledající lepší řešení jedince pouze v jeho sousedství. Při své práci algoritmus vygeneruje náhodně ve svém nejbližším okolí nové jedince, ze kterých následně vybere nejlepšího. Pokud tento nejlepší jedinec má lepší vhodnost než původní jedinec, tak toho jedince nahradí v další generaci. Nejbližším okolím je myšleno sférické okolí o průměru desetiny velikosti prohledávané funkce, což je také maximální možná vzdálenost generace potomků v nejbližším okolí. Jedinci díky tomuto pravidlu většinou skončí v lokálních extrémech. Více se tímto algoritmem zabývá práce „*Hill-Climbing search*“ [28]. V projektu je především, protože jej využívá další algoritmus pro lokální optimalizaci během evoluce, ale je možno jej využít i samostatně.

4.2.1. Třídní diagram

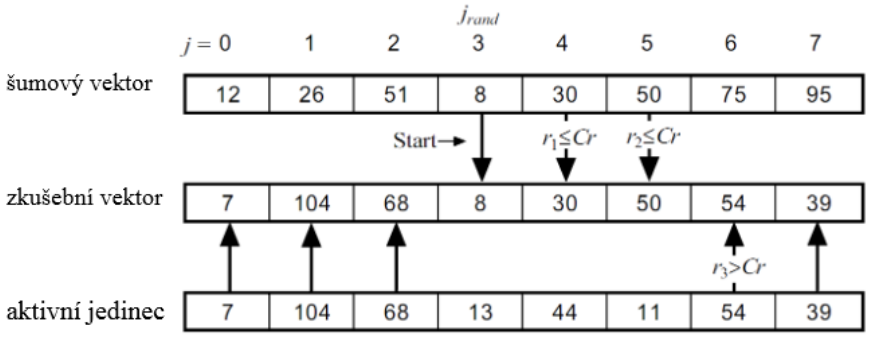


Obrázek 16 Třídní diagram - Climber

4.3. Diferenciální evoluce

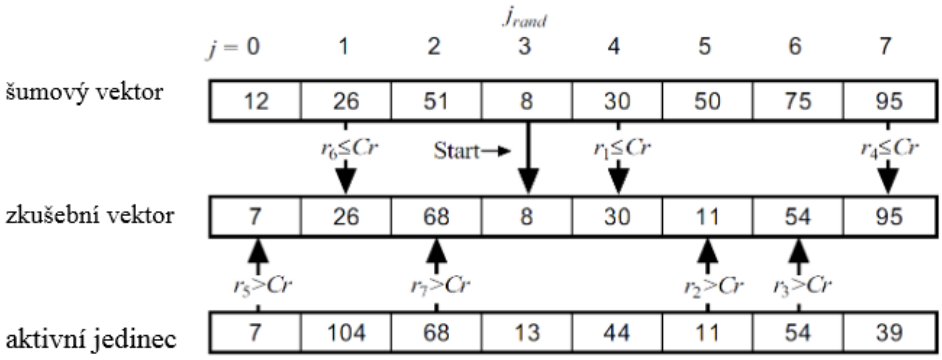
Principem diferenciálních algoritmů se zabývá práce *„Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces"*[33]. V knihovně je vytvořeno 5 typů diferenciálních evolucí. Každý je navíc ve verzi binomické (bin) a exponenciální (exp).

Rozdíl mezi verzi bin a exp je v tom, že u exponenciálního křížení se vybere parametr jedince, od kterého začne křížení. Tento parametr se zkopíruje z šumového vektoru do zkušebního a dále se standardně generuje náhodné číslo, které se porovná s prahem křížení. Dokud je náhodná hodnota menší nebo rovna prahu křížení, přebírá zkušební vektor hodnoty z vektoru šumového. V momentě, kdy je náhodná hodnota větší než práh křížení, se zbytek hodnot zkušebního vektoru přebere z aktivního jedince [3], viz Obrázek 17 Exponenciální křížení.



Obrázek 17 Exponenciální křížení

Binomické křížení také náhodně vybere první parametr jedince a zkopíruje jeho hodnotu, dále se však pro každý parametr generuje náhodné číslo, které se porovnává s prahem křížení. Pokud je menší nebo rovno, použije se parametr z šumového vektoru; v opačném případě se použije parametr z aktivního jedince. [3]. Viz Obrázek 18 Binomické křížení.



Obrázek 18 Binomické křížení

Dále je v knihovně implementováno 6 novějších typů diferenciálních evolucí, konkrétně „*Composite differential evolution*“ (CODE) [29], „*Adaptive differential evolution*“ (jDE) [30], „*Intersect mutation differential evolution*“ (IMDE) [31], „*Hybrid self-adaptive differential evolution*“ (HSDE) [32], „*Adaptive differential evolution with optional archive*“ (JADE) [18] a „*self-adaptive differential evolution*“ (SADE) [19,20]. Podrobnější popis je uveden přímo u konkrétní kapitoly s daným algoritmem.

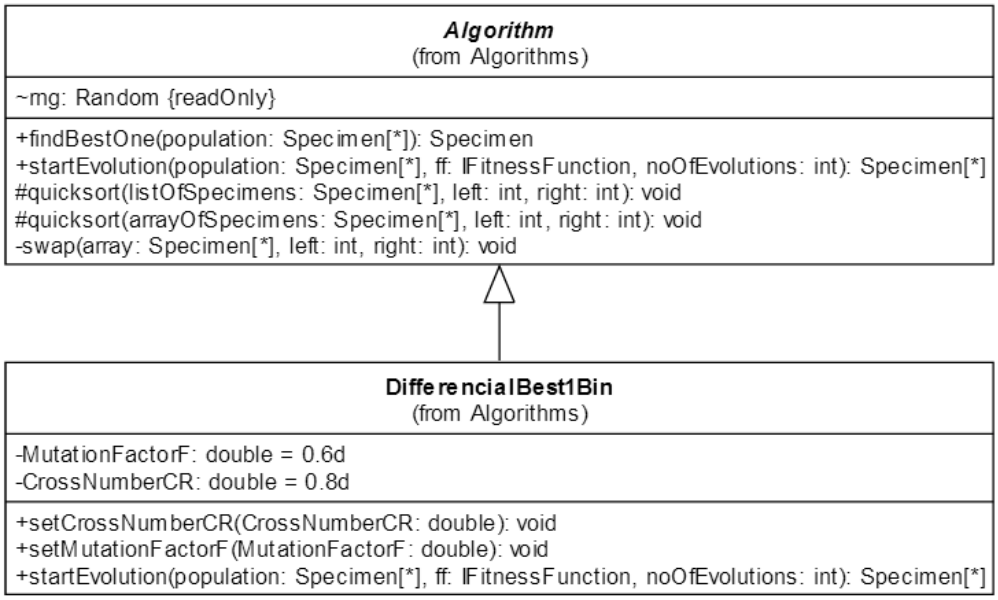
4.3.1. DE/Best/1/Bin

Diferenciální evoluce, která pro výpočet šumového vektoru využívá dva náhodné rodiče v populaci a jedince, který má v aktuální generaci nejlepší vhodnost. Patří mezi jednu z prvních strategií navrženou Pricem a Stronem. Šumový vektor se počítá z následující rovnice (1).

$$v_i = x^G_{best,j} + F * \left(x^G_{r1,j} - x^G_{r2,j}\right) \quad (1)$$

v_i… šumový vektor
$x^G_{best,j}$… jedinec s nejlepší vhodností v aktuální generaci
$x^G_{r1,j}$… první náhodně vybraný jedinec z generace
$x^G_{r2,j}$… druhý náhodně vybraný jedinec z generace
F … mutační konstanta

4.3.1.1. Třídní diagram

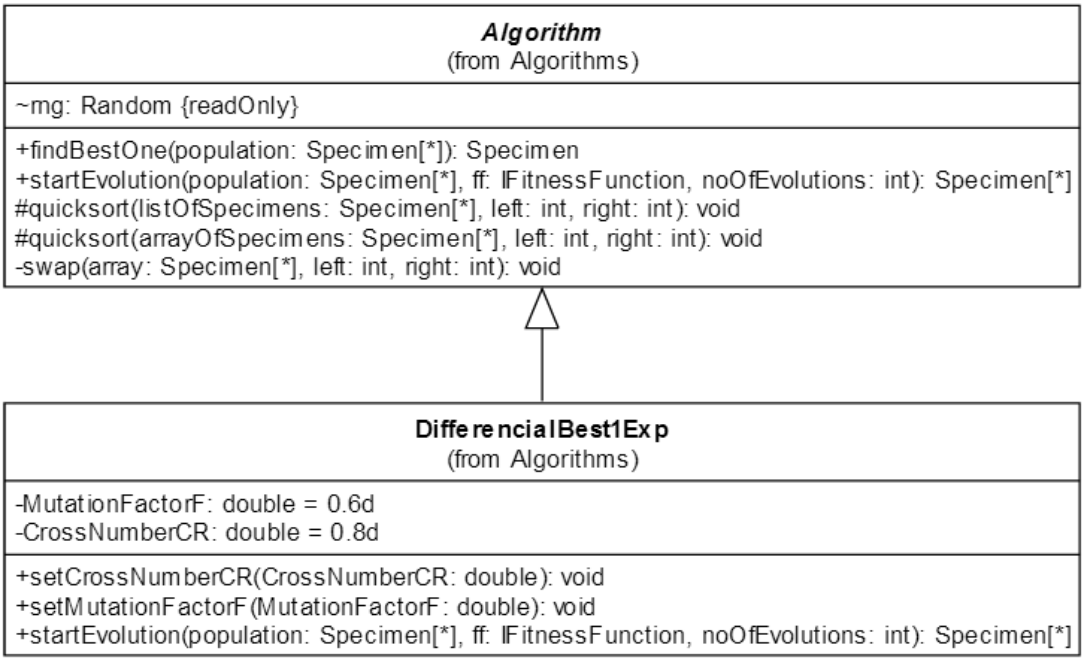


Obrázek 19 Třídní diagram - Best1Bin

4.3.2. DE/Best/1/Exp

Exponenciální verze strategie, která je stejná jako evoluce popsána v kapitole 4.3.1, jen pro křížení se používá exponenciální křížení, jak je vysvětleno v kapitole 4.3. Rovnice pro výpočet šumového vektoru zůstává stejná (1).

4.3.2.1. Třídní diagram



Obrázek 20 Třídní diagram - Best1Exp

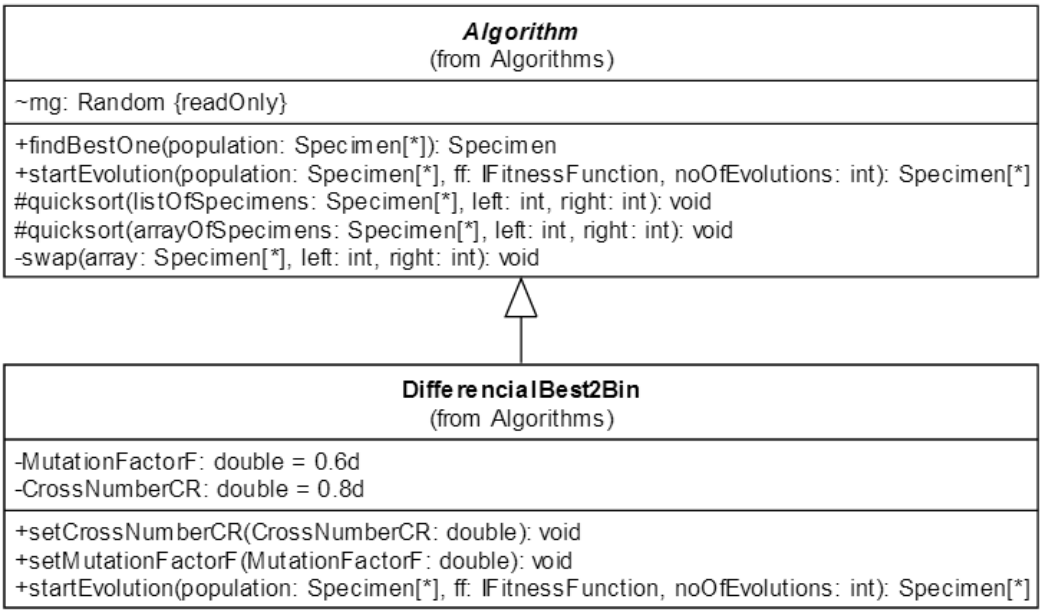
4.3.3. DE/Best/2/Bin

Další z prvních strategií navrhnutých Pricem a Stornem. Tato strategie pro výpočet šumového vektoru používá čtyři náhodně vybrané jedince v kombinaci s nejlepším jedincem v generaci. Tento výpočet je znázorněn v rovnici (2).

$$v_i = x^G_{best,j} + F * (x^G_{r1,j} + x^G_{r2,j} - x^G_{r3,j} - x^G_{r4,j}) \quad (2)$$

v_i… šumový vektor
$x^G_{best,j}$… jedinec s nejlepší vhodností v aktuální generaci
$x^G_{r1,j}$… první náhodně vybraný jedinec z generace
$x^G_{r2,j}$… druhý náhodně vybraný jedinec z generace
$x^G_{r3,j}$… třetí náhodně vybraný jedinec z generace
$x^G_{r4,j}$… čtvrtý náhodně vybraný jedinec z generace
F… mutační konstanta

4.3.3.1. Třídní diagram

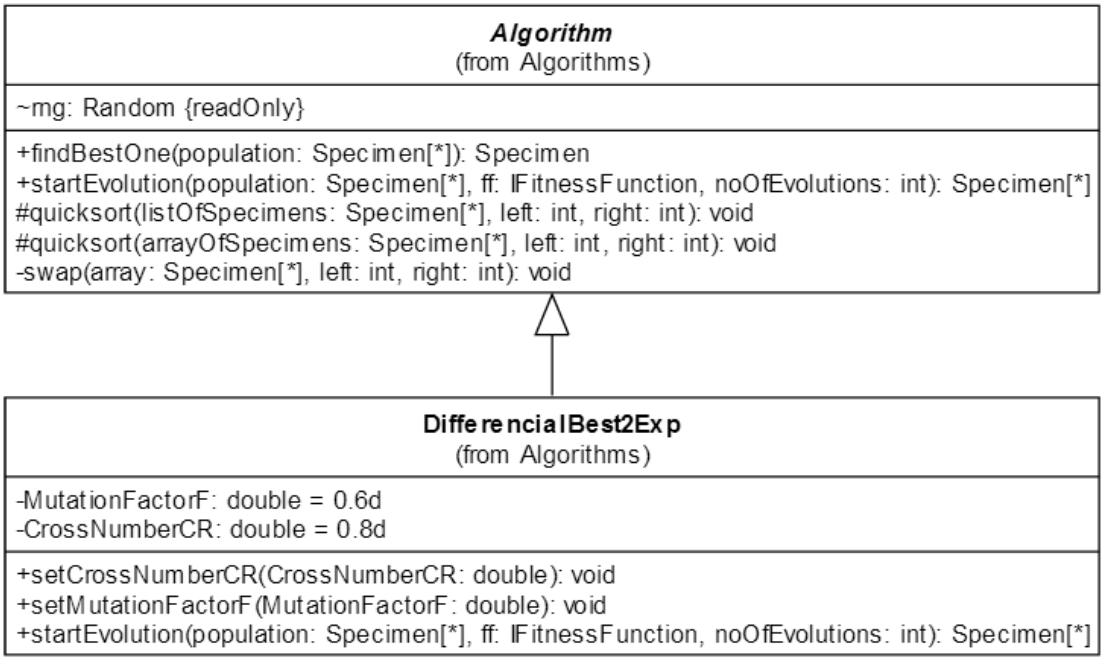


Obrázek 21 Třídní diagram - Best2Bin

4.3.4. DE/Best/2/Exp

Strategie, která se oproti kapitole 4.3.3 liší pouze v použití exponenciálního křížení, jak je vysvětleno v kapitole 4.3. Rovnice pro výpočet šumového vektoru se nemění (2).

4.3.4.1. Třídní diagram



Obrázek 22 Třídní diagram - Best2Exp

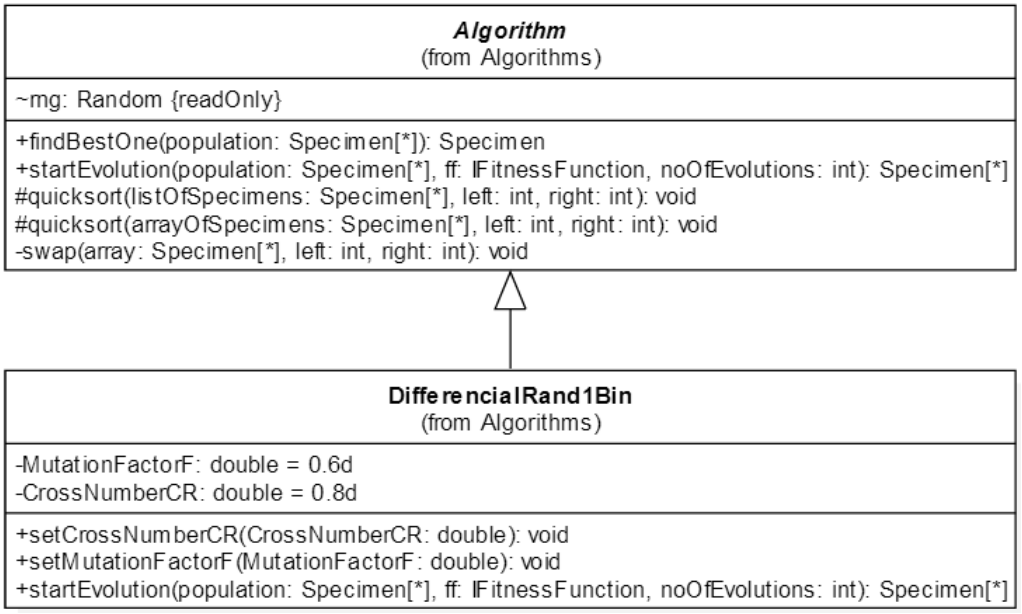
4.3.5. DE/Rand/1/Bin

Diferenciální evoluce, ve které jsou všichni rodiče vybíráni zcela náhodně bez ohledu na jejich vhodnost. Výpočet šumového vektoru je popsán v rovnici (3). Opět je strategie navrhnuta jako jedna z prvních Pricem a Stornem.

$$v_i = x^G_{r1,j} + F * (x^G_{r2,j} - x^G_{r3,j}) \quad (3)$$

v_i… šumový vektor
$x^G_{r1,j}$… první náhodně vybraný jedinec z generace
$x^G_{r2,j}$… druhý náhodně vybraný jedinec z generace
$x^G_{r3,j}$… třetí náhodně vybraný jedinec z generace
F… mutační konstanta

4.3.5.1. Třídní diagram

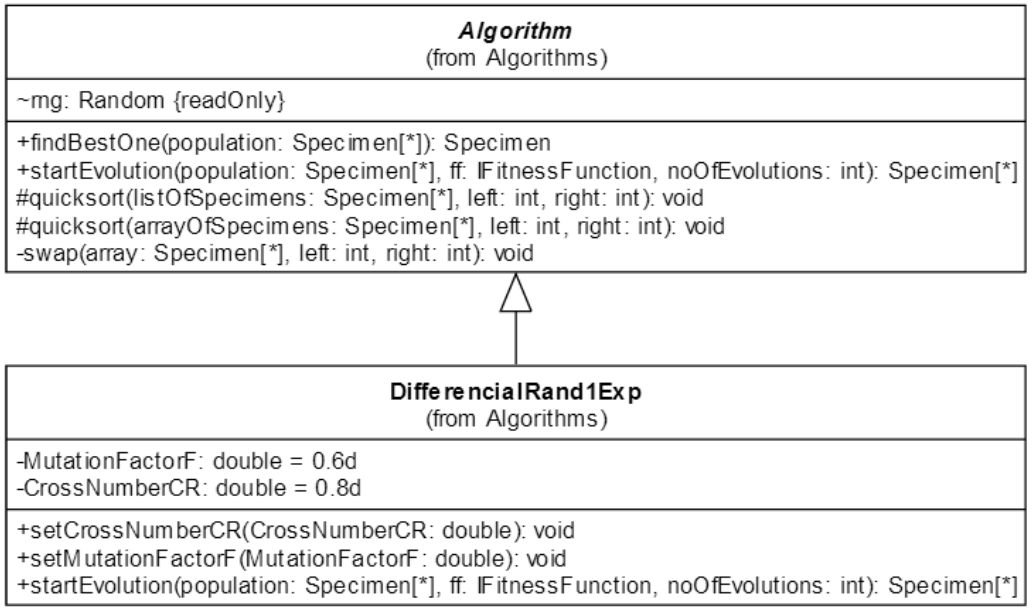


Obrázek 23 Třídní diagram - Rand1Bin

4.3.6. DE/Rand/1/Exp

Exponenciální verze strategie, která je popsána v kapitole 4.3.5, jen pro křížení používá exponenciální křížení, jak je vysvětleno v kapitole 4.3.

4.3.6.1. Třídní diagram



Obrázek 24 Třídní diagram - Rand1Exp

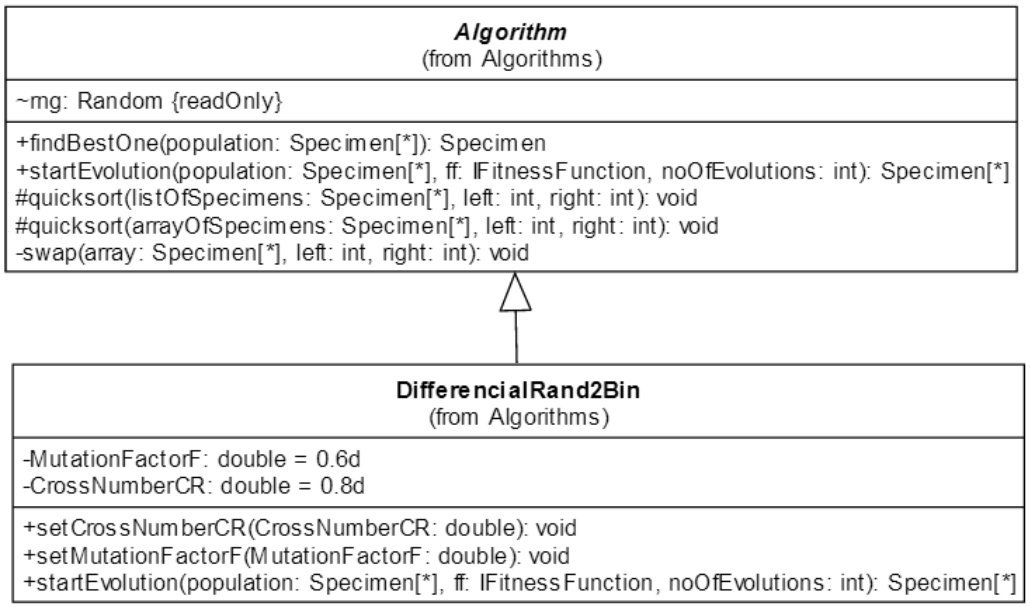
4.3.7. DE/Rand/2/Bin

Varianta podobná variantě DE/Best/2/Bin, akorát jedinec s nejlepší vhodností v populaci byl nahrazen dalším náhodně vybraným rodičem. Předpis rovnice (4). Strategie také patří k prvním strategiím, které popsali Price a Storn.

$$v_i = x^G_{r1,j} + F * (x^G_{r2,j} + x^G_{r3,j} - x^G_{r4,j} - x^G_{r5,j}) \quad (4)$$

v_i… šumový vektor
$x^G_{r1,j}$… první náhodně vybraný jedinec z generace
$x^G_{r2,j}$… druhý náhodně vybraný jedinec z generace
$x^G_{r3,j}$… třetí náhodně vybraný jedinec z generace
$x^G_{r4,j}$… čtvrtý náhodně vybraný jedinec z generace
$x^G_{r5,j}$… pátý náhodně vybraný jedinec z generace
F… mutační konstanta

4.3.7.1. Třídní diagram

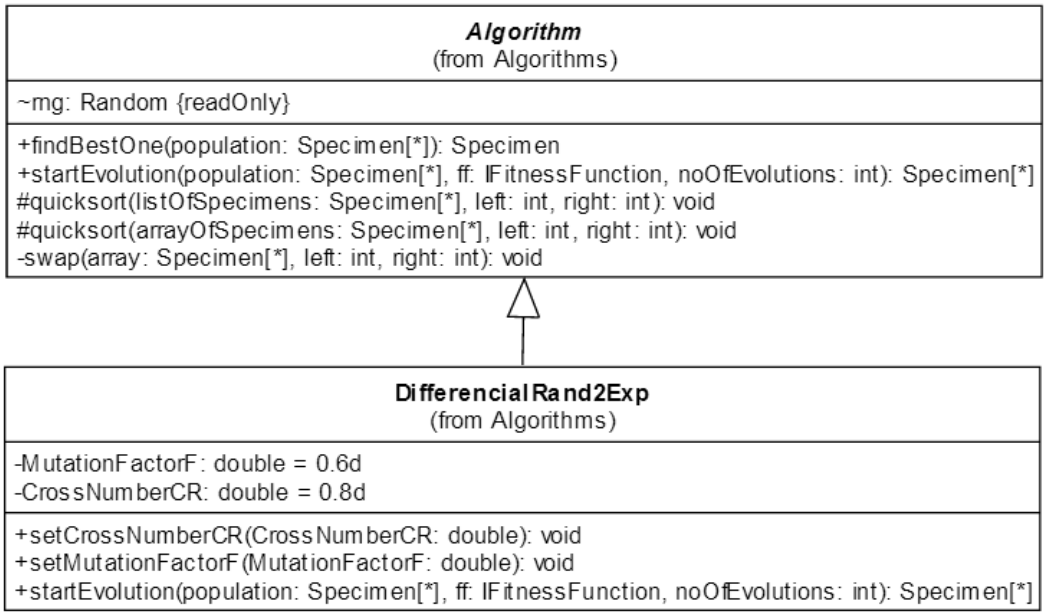


Obrázek 25Třídní diagram - Rand2Bin

4.3.8. DE/Rand/2/Exp

Exponenciální verze strategie pěti náhodně vybranými rodiči, jak je popsáno v kapitole 4.3.7, princip exponenciálního křížení je vysvětlen v kapitole 4.3.

4.3.8.1. Třídní diagram



Obrázek 26 Třídní diagram - Rand2Exp

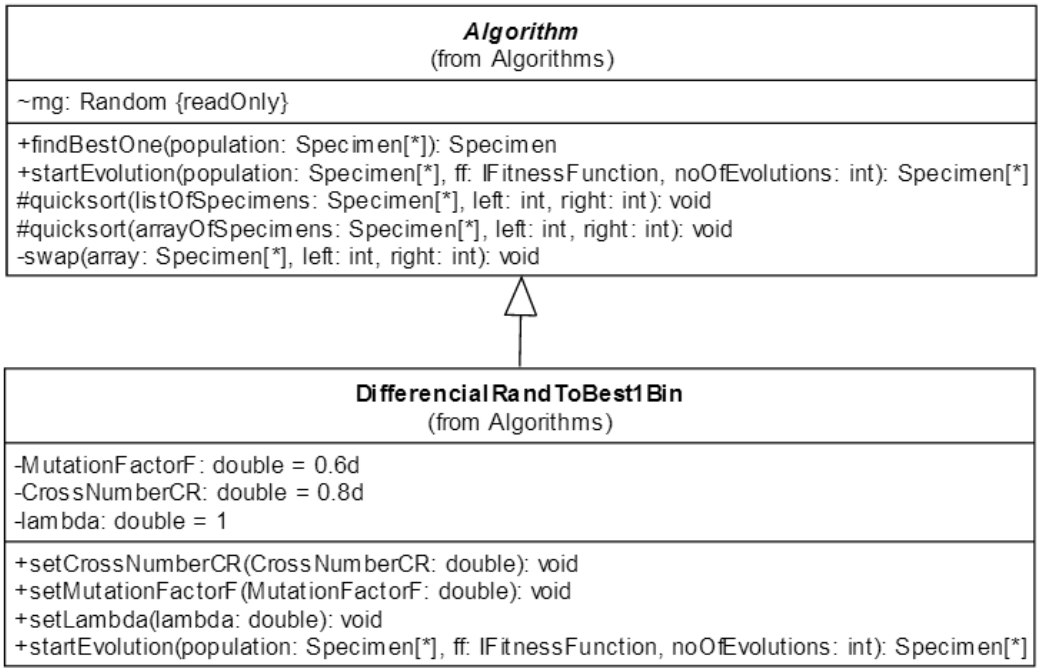
4.3.9. DE/RandToBest/1/Bin

Tato verze oproti ostatním obsahuje proměnnou λ, která funguje stejně jako mutační konstanta, ale pouze pro rozdíl vzdálenosti mezi aktuálním a nejlepším jedincem v populaci. Defaultně je tato hodnota nastavena na 1, takže rozdíl mezi nejvhodnějším a aktuálním jedincem není nijak upravován, jak lze vidět v rovnici (5). Nastavením proměnné λ na hodnotu 0 v podstatě odstraníme ze vzorce rozdíl mezi aktuálním a nejvhodnějším jedincem, čímž dostaneme strategii DE/current/1Bin. Tuto strategii také navrhnuli zakladatelé diferenciální evoluce Price a Storn.

$$v_i = x_{i,j}^G + \lambda * \left(x_{best,j}^G - x_{i,j}^G\right) + F * (x_{r1,j}^G - x_{r2,j}^G) \quad (5)$$

v_i… šumový vektor
$x_{i,j}^G$…jedinec s nejlepší vhodností v aktuální generaci
$x_{best,j}^G$…nejlepší jedinec z generace
$x_{r1,j}^G$… první náhodně vybraný jedinec z generace
$x_{r2,j}^G$… druhý náhodně vybraný jedinec z generace
λ … parametr lambda
F… mutační konstanta

4.3.9.1. Třídní diagram

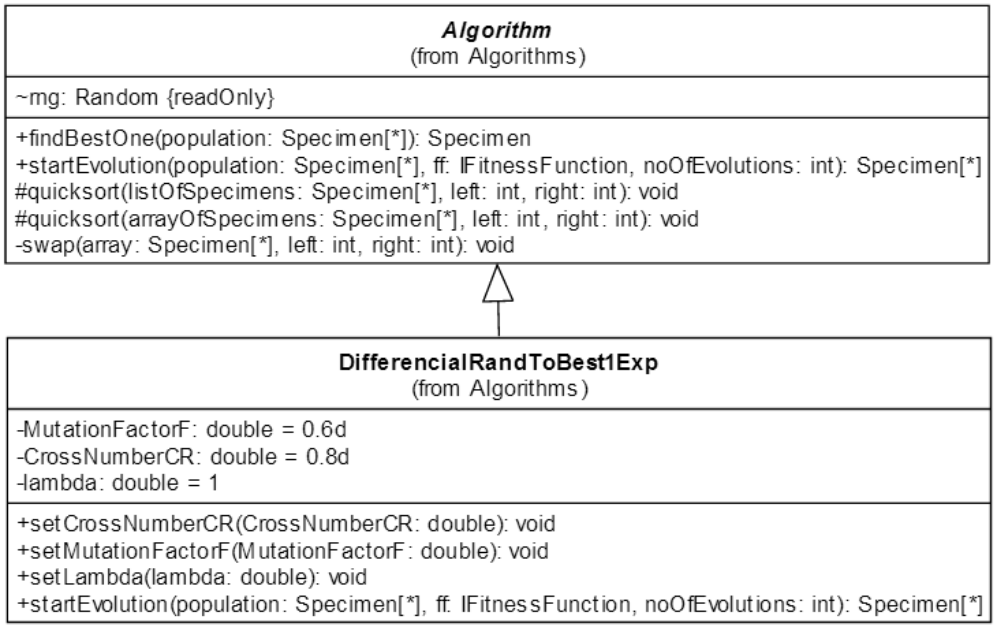


Obrázek 27 Třídní diagram - RandToBest1Bin

4.3.10. DE/RandToBest/1/Exp

Verze diferenciální evoluce, která je stejná jako verze popsána v kapitole 4.3.9, jen pro křížení používá exponenciální křížení, jak je vysvětleno v kapitole 4.3. V případě nastavení proměnné λ na hodnotu 0 dostaneme strategii DE/current/1/Exp.

4.3.10.1. Třídní diagram



Obrázek 28 Třídní diagram - RandToBest1Exp

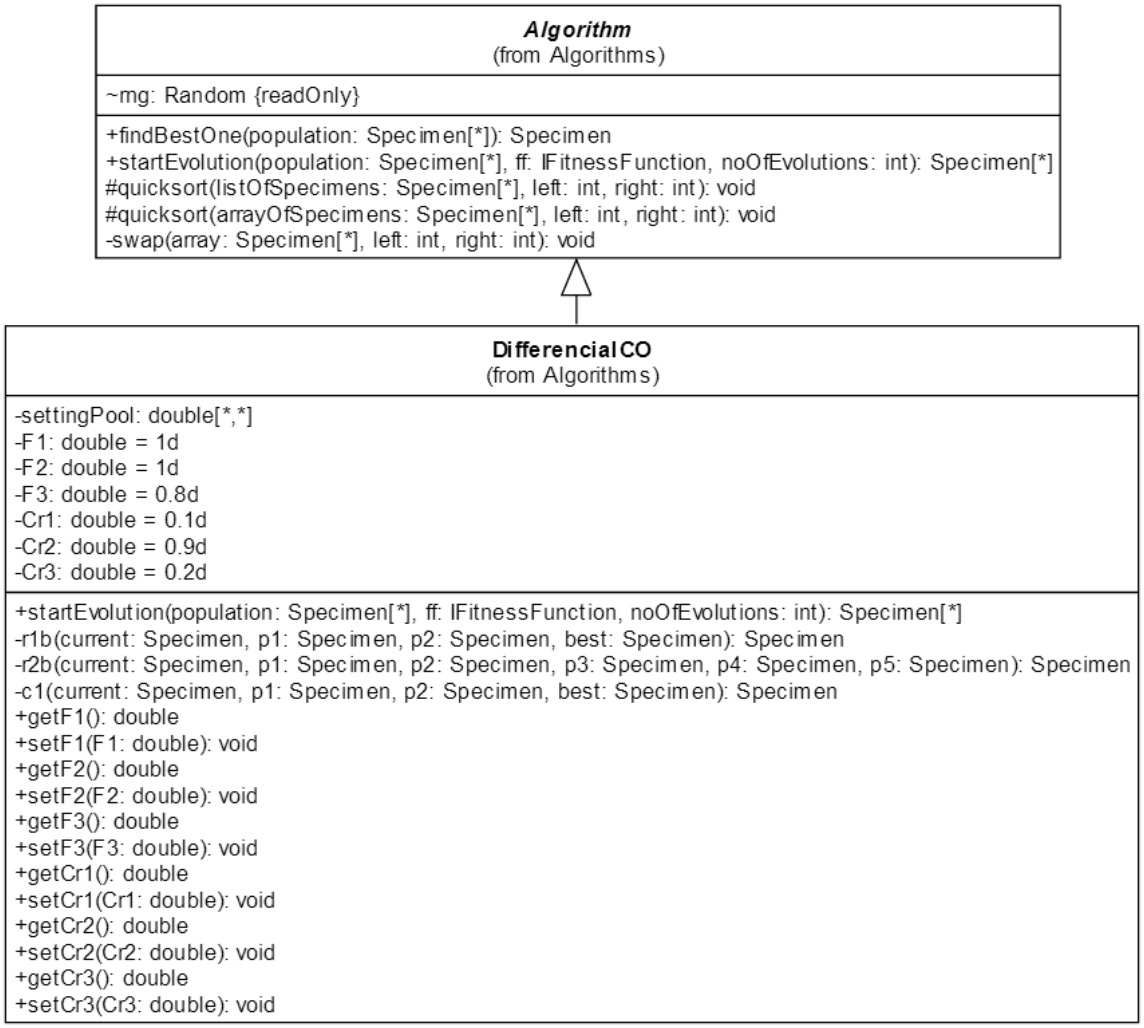
4.3.11. CODE

Kompozitní verze diferenciální evoluce, která pro svou práci využívá postupné volání tří typů diferenciálních evolucí a to DE/rand/1/bin, DE/rand/2/bin a DE/current-to-best/1. Je vhodné podotknout, že ve strategii DE/current-to-best/1 nedochází k žádnému z výše uvedenému typu křížení, ale kříží se vždy a vše. Na každého jedince v populaci se aplikují všechny tři strategie, jejichž parametry jsou náhodně vybírány z množiny kandidátů. Takto jsou získány tři šumové vektory. Z těch se vybere nejvhodnější vektor, který pak standardně postupuje dále ve výběru [29].

Kontrolní parametry jsou definovány v množině kandidátů, která je ve třídě inicializována jako pole. Každý parametr lze uživatelsky změnit. Obsah množiny kandidátů je v základním nastavení následující.
1) [F= 1.0,C_r= 0.1]
2)[F= 1.0,C_r= 0.9]
3) [F= 0.8,C_r= 0.2]

Tento algoritmus byl využit například v pracích „*Optimal synthesis of linear antenna array with composite differential evolution algorithm*“, kde pomocí CODE dosáhli velmi efektivního nastavení antény oproti nastavení pomocí strategií DE, jDE, SADE, JADE, BBP, GA, PSO nebo ES [44], nebo „*Application of a Composite Differential Evolution Algorithm in Optimal Neural Network Design for Propagation Path-Loss Prediction in Mobile Communication Systems*“ ve které byly získány lepší výsledky pro návrh umělé nervové sítě, než pomocí v té době populární strategie jDE[45].

4.3.11.1. Třídní diagram



Obrázek 29 Třídní diagram –CODE

4.3.12. jDE

Adaptivní verze diferenciální evoluce, která využívá kombinaci tří evolucí (DE/best/1/bin, DE/rand/1/bin, DE/rand/1/exp) ze kterých se vybírá dle rovnice (6). DE/best/1/bin má navíc 10% šanci, že bude použit kdykoliv. [30] Vybrané tři evoluce se mohou v každé práci lišit dle potřeb konkrétního problému. Právě proto se stala tato strategie velmi oblíbená. Po každé evoluci je šance, že se změní řídící parametry křížení a mutace, dle rovnice (7) [30].

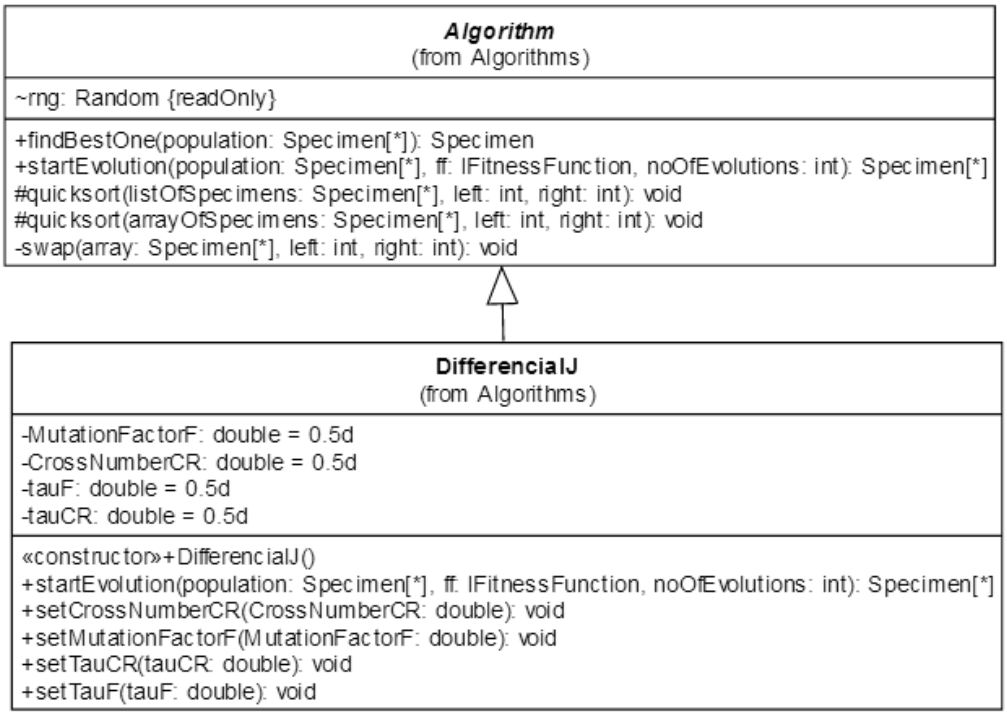
$$\begin{cases} if\ gen_i > \frac{gen_{max}}{2} & ,than\ \ DE/best/1/bin \\ if\ (gen_i\ mod\ (popSize)) < \frac{popSize}{2} & ,than\ DE/rand/1/bin \\ else & DE/rand/1/exp \end{cases} \quad (6)$$

gen_i …aktuální generace
gen_{max} …maximální počet generací
$popSize$ …velikost populace

$$\begin{cases} if\ rand_1 < \tau_{CR} & ,than\ CR_{G+1} = rand_{CR} \\ if\ rand_2 < \tau_F & ,than\ F_{G+1} = rand_F \end{cases} \quad (7)$$

$rand_1$ … první náhodně vygenerované číslo
$rand_2$ … druhé náhodně vygenerované číslo
τ_{CR} … pravděpodpbnost změny práhu křížení
τ_F … pravděpodobnost změny mutační konstanty
CR_{G+1} … práh křížení pro další generaci
F_{G+1} … mutační konstanta pro další generaci
$rand_{CR}$ … náhodně vygenerovaný práh křížení v intervalu $\langle 0; 1)$
$rand_F$ … náhodně vygenerovaná mutační konstanta v intervalu $\langle 0; 1)$

4.3.12.1. Třídní diagram



Obrázek 30 Třídní diagram - jDE

4.3.13. IMDE

Tato verze diferenciální evoluce je rozdělena na dva procesy. Nejprve se jedinci rozdělí na lepší a horší jedince. Následně proběhnou dva procesy evoluce. První proces stojí na strategii DE/rand/1/bin a díky plnění rovnice z lepších a horších jedinců se eliminuje slabost této strategie, automatické směřování k lepšímu řešení. Druhý proces je založený na strategii DE/current-to-best/1/bin, a opět, díky plnění rovnice této strategie lepšími a horšími jedinci, je eliminována její slabost při řešení multimodálních problémů [31]. Každý proces navíc plní rovnici jinak v závislosti na tom, jestli aktuální jedinec patří do lepší, nebo horší části. Rovnice pro první proces a aktuálním jedincem z lepší poloviny je (8), v případě, že je aktuální jedinec v horší polovině populace, použije se rovnice (9). V druhém procesu se použije rovnice (10) v případě, že je jedinec v lepší polovině populace, jinak se použije rovnice (11)

$$v_i = x_{r1}^{Glow} + F * (x_{r1}^{Gtop} - x_{r2}^{Gtop}) \quad (8)$$

$$v_i = x_{r1}^{Gtop} + F * (x_{r1}^{Glow} - x_{r2}^{Glow}) \quad (9)$$

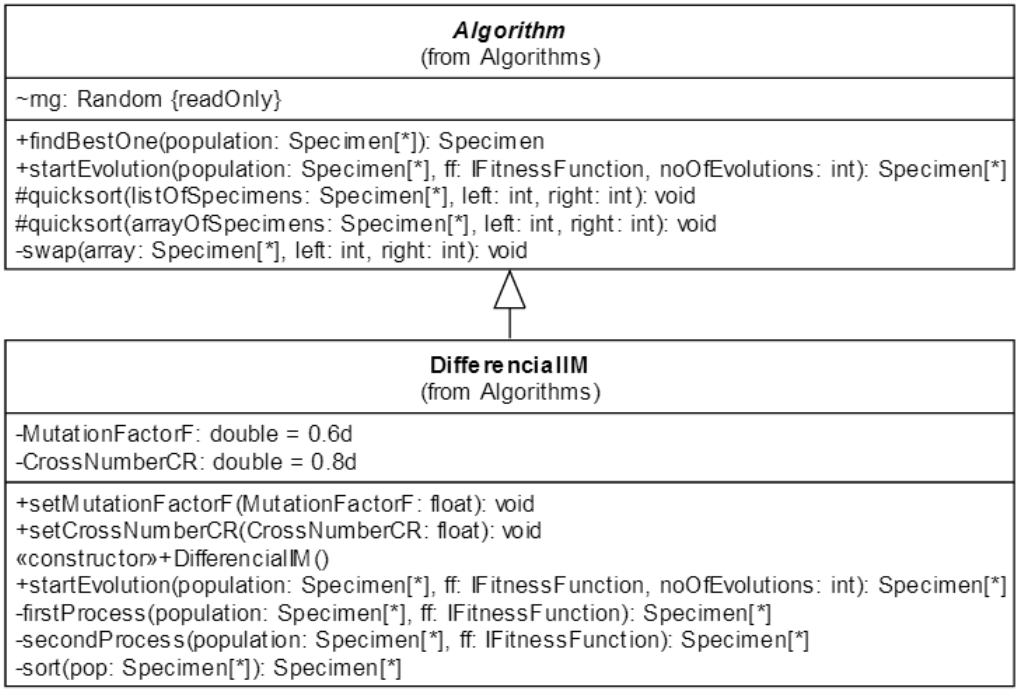
$$v_i = x_{r1}^{Glow} + F * (x_{r2}^{Glow} - x_{r1}^{Gtop}) \quad (10)$$

$$v_i = x_{r1}^{Gtop} + F * (x_{r1}^{Glow} - x_{r2}^{Glow}) \quad (11)$$

v_i… šumový vektor
x_{r1}^{Glow}… náhodně vybraný jedinec z populační části s horší vhodností
F … mutační konstanta
x_{r1}^{Gtop}… první náhodně vybraný jedinec z populační části s lepší vhodností
x_{r2}^{Gtop}… druhý náhodně vybraný jedinec z populační části s lepší vhodností
v_i… šumový vektor
x_{r1}^{Glow}… první náhodně vybraný jedinec z populační části s horší vhodností
x_{r2}^{Glow}… druhý náhodně vybraný jedinec z populační části s horší vhodností
F … mutační konstanta
x_{r1}^{Gtop}… první náhodně vybraný jedinec z populační části s lepší vhodností
x_{r2}^{Gtop}… druhý náhodně vybraný jedinec z populační části s lepší vhodností

Strategie právě díky rozdělení populace prokazuje kvalitnější výsledky než tradiční diferenční evoluce, například když byl použit [31] na 30 dimenzionální problém a zvažují se možnosti, jak tento typ evoluce zakomponovat do dalších strategií tak, aby došlo k jejímu vylepšení, například v kombinaci s automatickým nastavením parametrů [31].

4.3.13.1. Třídní diagram



Obrázek 31 Třídní diagram - IMDE

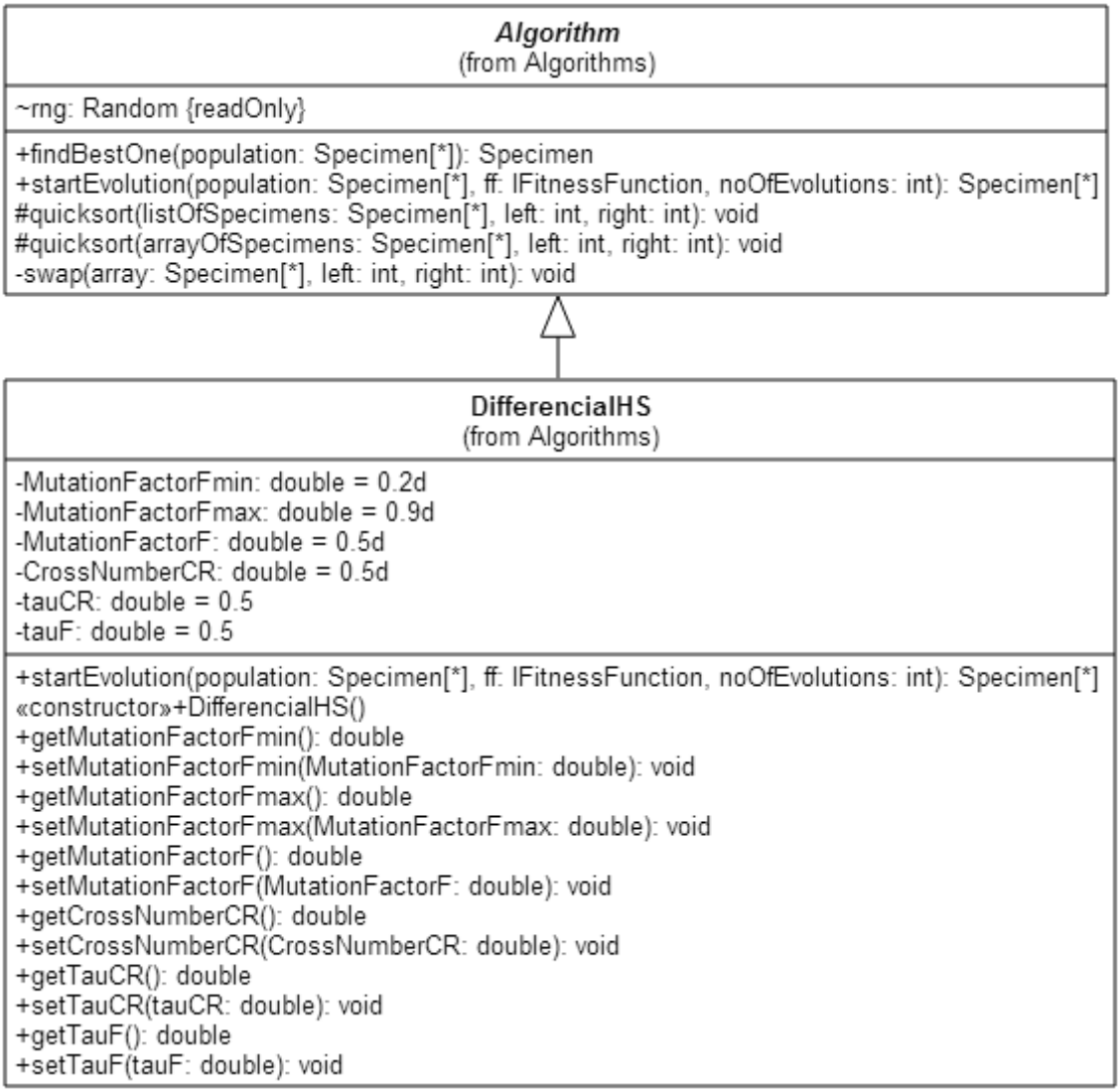
4.3.14. HSDE

Hybridní, sebe-adaptivní diferenciální evoluce využívá evoluci DE/rand/1/bin a v průběhu evoluce mění její parametry – mutační konstanta a práh křížení. Tyto parametry se pro každou generaci změní

s pravděpodobností τ, která je v základním nastavení rovna 50%. Generování nové mutační konstanty můžeme ovlivnit nastavením maximální a minimální hodnoty, mezi kterou se má generovat. Práh křížení žádné omezení nemá a může se rovnat hodnotám od nuly do jedné, tedy od 0% do 100% [32].

Evoluce byla využita v práci "*An Effective Hybrid Self-Adapting Differential Evolution Algorithm for the Joint Replenishment and Location-Inventory Problem in a Three-Level Supply Chain*" [48], pro řešení NP-těžkého problému, také v této práci byla poprvé použita některá z vylepšených diferenciálních evolucí pro řešení tohoto NP-těžkého problému, se zjištěním že HSDE tento problém snadno vyřeší. Navíc v této práci zjistili, že strategie je vhodnější pro větší počet dimenzí poté, co v benchmarku vycházely lepší výsledky pro 30 dimenzí, oproti testu na 10 dimenzích [48].

4.3.14.1. Třídní diagram



Obrázek 32 Třídní diagram - HSDE

4.3.15. JADE

Adaptivní diferenciální evoluce s volitelným archivem využívá evoluci DE/current-to-pbest, kde *pBest* je vrchních několik procent nejlepších jedinců v aktuální populaci, čímž se zabrání rychlému přesunu

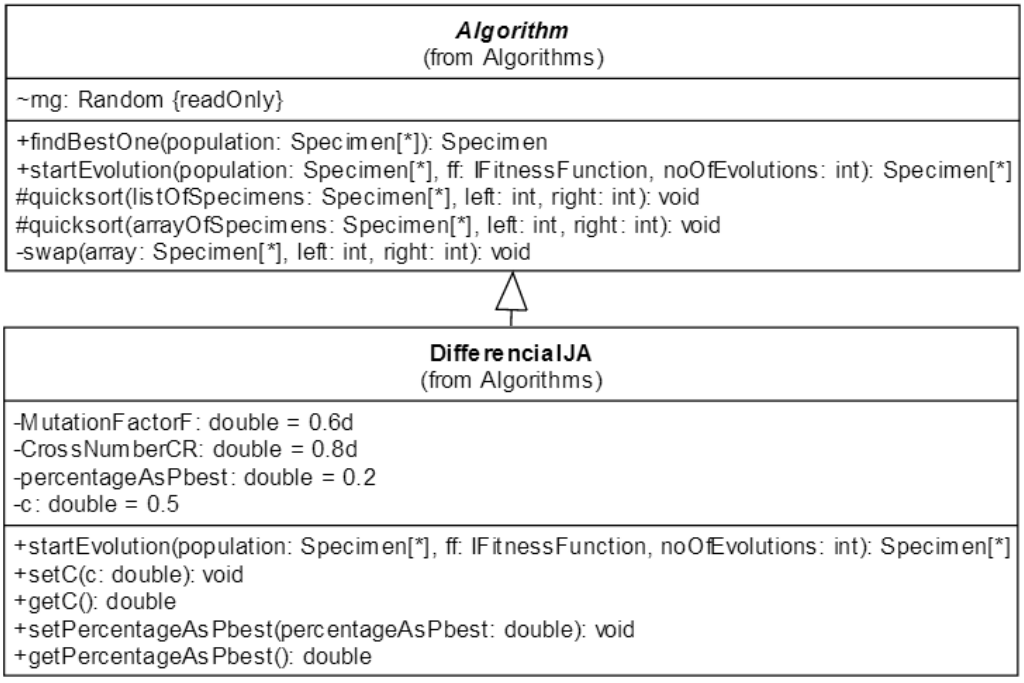
k nejlepšímu jedinci v populaci. Procentuální zastoupení nejlepších jedinců se volí parametrem *percentageAsPbest*, jehož základní hodnota je 0,2. Což znamená, že jako *pBest* se vybere náhodně jeden z 20% nejlepších jedinců v aktuální generaci. Poté co proběhne evoluce, se každý vyřazený jedinec (ať už rodič nebo nový, nevyhovující potomek) archivuje ve speciálním poli zvaném archiv. Archiv uchová maximálně tolik jedinců, kolik jedinců je v aktivní populaci, aby se předešlo případným problémům s pamětí. V případě, že v archivu dojde místo, je z něj náhodně odstraněn libovolný jedinec. Pomocí archivu mohou vyřazení jedinci ovlivnit další evoluci, tím pádem není nevyhovující řešení okamžitě zavrhnuto. Nakonec je po každé generaci náhodně generována nová mutační konstanta *F* podle Cauchyho rozdělení a práh křížení *Cr* podle normálového rozdělení se střední hodnotou založenou na základě úspěšnosti použití těchto faktorů a standartní odchylkou 0,1. [18]. Do knihovny byl zažazen právě díky jeho schopnosti práce s nevyhovujicími jedinci, místo toho aby je automaticky zavrhl jako ostatní diferenciální evoluce.

$$v = x_{i,j}^{G} + F_i * \left(x_{pBest,j}^{G} - x_{i,j}^{G}\right) + F_i * (x_{r1,j}^{G} - x_{ra1,j}^{G}) \quad (12)$$

v… šumový vektor
$x_{i,j}^{G}$… aktuálně porovnávaný jedinec z generace
$x_{pBest,j}^{G}$… nejlepší jedinec náhodně vybraný mezi několika nejlepšími jedinci v generaci
$x_{r1,j}^{G}$… náhodně vybraný jedinec z generace
$x_{ra2,j}^{G}$… náhodně vybraný jedinec ze sjednocení množin archivu a aktuální populace
F_i… mutační konstanta pro danou generaci

JADE byl použit jako jeden z evolučních strategií v práci „*Evolving Artificial Neural Networks Using Adaptive Differential Evolution*“ [49], kde se prokázal jako nejlepší algoritmus pro výběr nejlepší umělé nervové sítě.

4.3.15.1. Třídní diagram



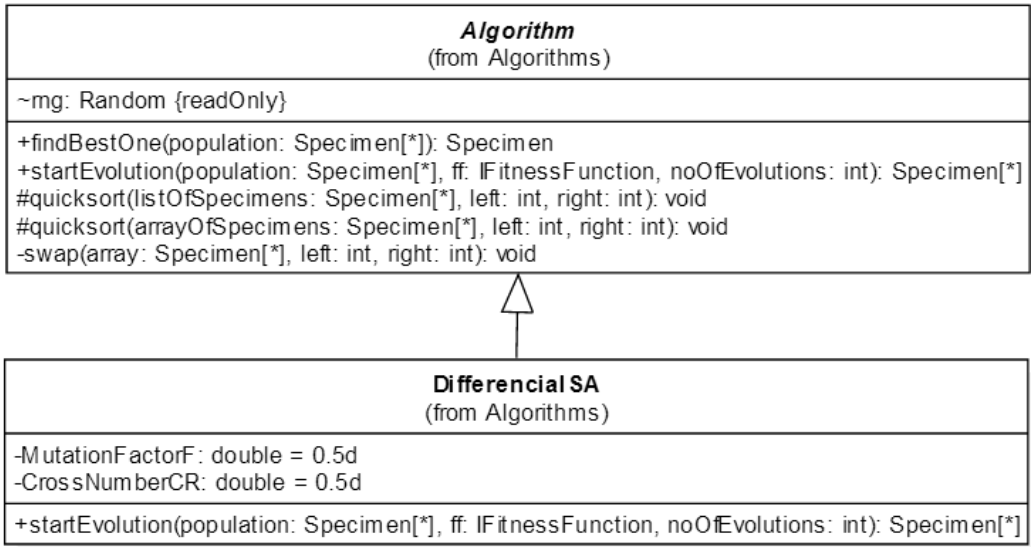
Obrázek 33 Třídní diagram - JADE

4.3.16. SADE

Samoadaptivní diferenciální evoluce využívá strategie DE/rand/1/bin pro její dobrou diverzitu a DE/current-to-best/2/bin pro její dobrou konvergenci. Pravděpodobnost, která z těchto strategií bude pro daného jedince použita, je náhodná s pravděpodobností *p*, která se po každé dokončené generaci přepočítává na základě toho, kolik nových jedinců prošlo do další generace díky které strategii. Navíc se každou generací mění mutační konstanta F, a co pět generací se mění střední hodnota prahu křížení na základě úspěšnosti využití křížení při tvorbě nových jedinců. Každý jedinec má svůj vlastní práh křížení [19,20].

Tento algoritmus byl použit pro řešení problému v práci „*Self-Adaptive Differential Evolution Applied to Real-Valued Antenna and Microwave Design Problems*“, kde jej aplikovali na běžné elektromagnetické problémy. Výsledek nebyl nijak zvlášť přesnější než použití jiných diferenčních evolucí, ale projevila se zde výhoda automatického nastavení konstanty křížení a prahu mutace, což zajistilo jeho snadnější použití na elektromagnetické problémy [46].

4.3.16.1. Třídní diagram



Obrázek 34 Třídní diagram - SADE

4.4. Evoluční strategie

Principem evolučních strategií se zabývá práce „*Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms“* [23]. Tyto strategie vznikaly zároveň s diferenciálními evolucemi, ale jejich princip se liší. Evoluční strategie při své práci vybírají rodiče. Dle dané strategie se vytvoří nový potomek a ten je následně zmutován. V podstatě tedy neprobíhá žádná operace křížení mezi rodiči. V knihovně jsou implementovány dvě verze evolučních strategií, Dimerická a Rekombinační.

Pro výpočet odchylky (mutace) vlastnosti jedince normálovým rozdělením se standartní odchylkou byla použita externí třída jsc.distribution.normal [4], která je součástí statistických tříd v Java,

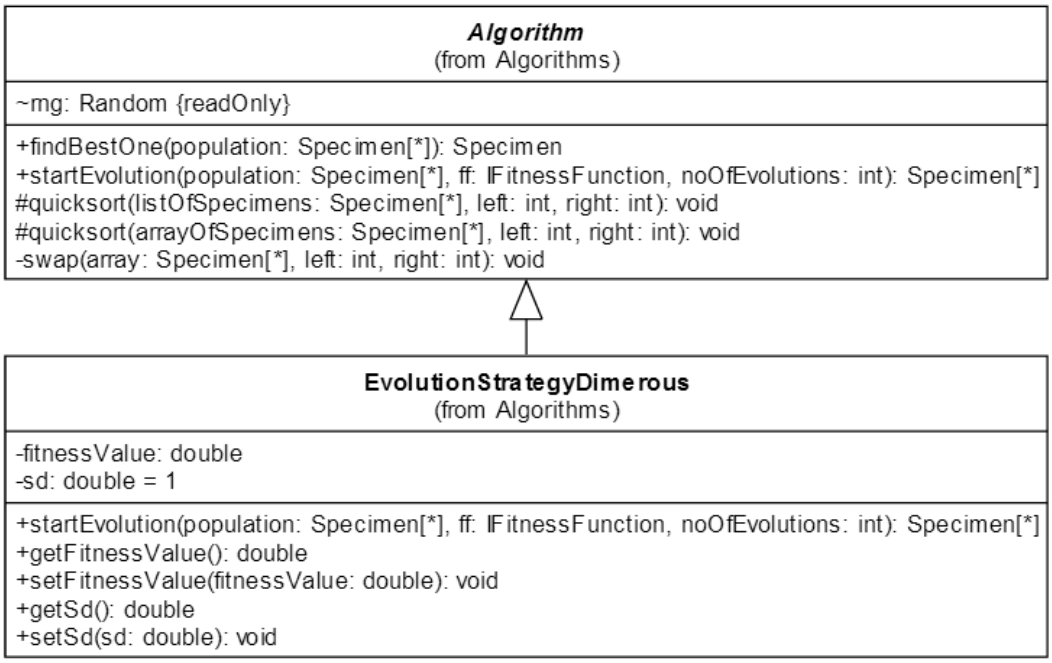
vytvořených pro podporu vzdělání a vývoje nového statistického software v Java [16]. Tuto odchylku lze nastavovat proměnnou *sd*, která je v základu nastavená na hodnotu 1.

Dále je zde proměnná *fitnessValue*, díky které se evoluce zastaví v momentě, kdy je nalezen jedinec s menší vhodností než proměnná. *FitnessValue* je defaultně nastavena na nejnižší možnou hodnotu třídy Double, konkrétně $2^{-1074[}$ [5].

4.4.1. Dimerická

Nejjednodušší verzí Evolučních strategie, která pomocí jednoho rodiče a Gaussovy odchylky vytváří nového jedince. Je-li vhodnost nového jedince lepší než vhodnost rodiče, je rodič potomkem nahrazen [23].

4.4.1.1. Třídní diagram



Obrázek 35 Třídní diagram - EvolutionStrategyDimerious

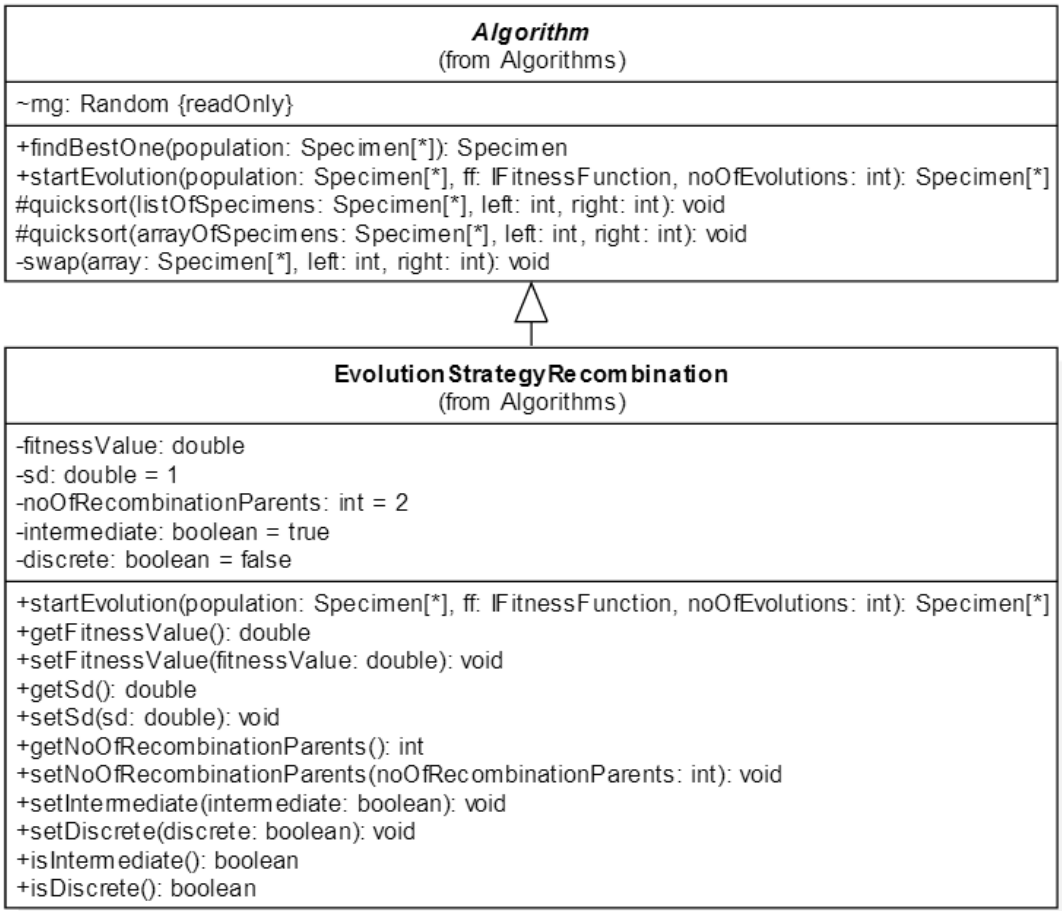
4.4.2. Rekombinační

Rekombinační verze umožnuje před vlastní mutací jedince vytvořit rekombinant několika rodičů, ze kterého se tvoří výsledný potomek. Počet rodičů určuje parametr *noOfRecombinationParents*, který je v základu nastaven na 2, tedy rekombinat se tvoří ze dvou náhodně vybraných rodičů [23].

Implementovanou rekombinační verzi je možno navíc nastavit na verzi intermediate a discrete, v základním nastavení je ve verzi intermediate. Nastavení verze probíhá v kódu přepnutím příslušné proměnné na true, což také automaticky nastaví druhou proměnnou do stavu false.

Verze intermediate, neboli průměrová verze, vytvoří průměr z parametrů rodičů a ten se následně mutuje. Tato verze je vhodná pro problematiku unimodálních funkcí. Diskrétní verze evoluce nejprve náhodně vybere rodiče, a poté potomkovi předává náhodné geny těchto rodičů. Potomek následně mutuje. [23]

4.4.2.1. Třídní diagram



Obrázek 36 Třídní diagram - EvolutionStrategyRecombination

4.5. Genetické algoritmy

Principem Genetických algoritmů se zabývá kniha „*Handbook of genetic algorithms"* [34] nebo kniha „*An introduction to genetic algorithms*" [35]. V knihovně jsou implementovány tři typy algoritmů: základní jednobodové křížení, hybridní a paralelní verze.

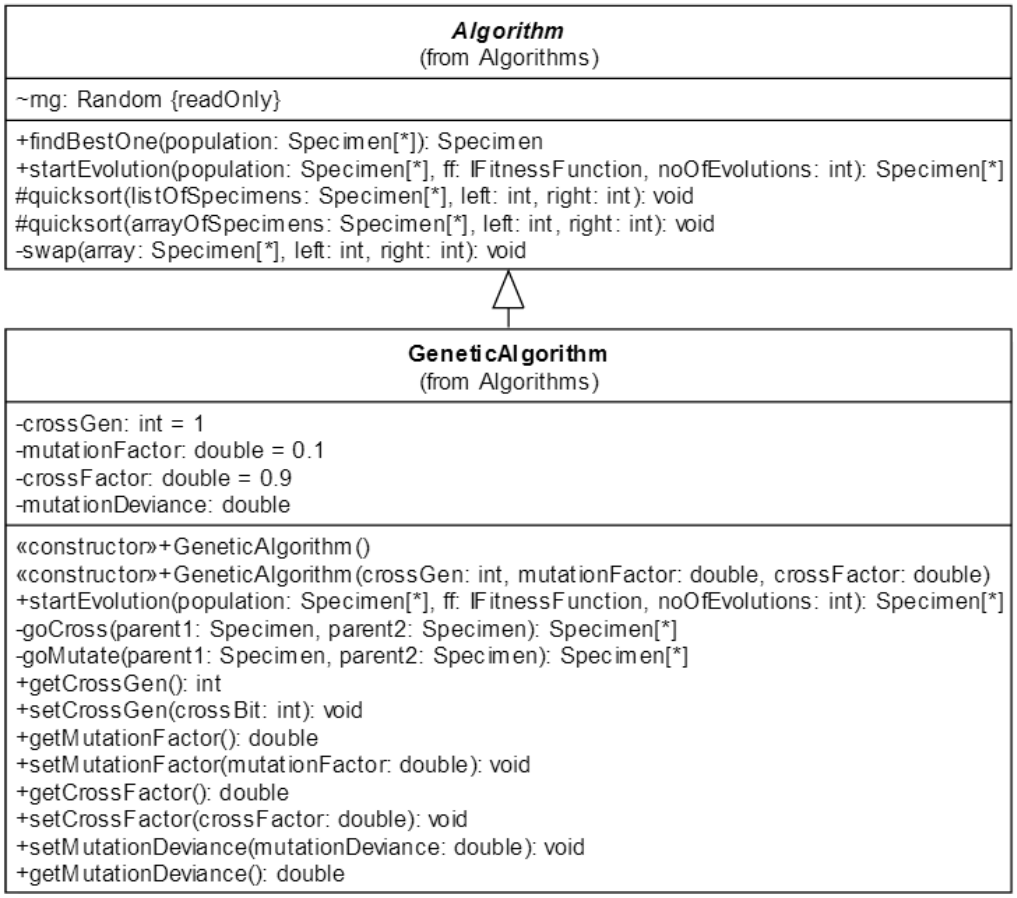
Všechny verze genetických algoritmů obsahují proměnnou *crossGen*, v základu nastavenou na hodnotu 1. Pokud uživatel nastaví vyšší hodnotu než je skutečný počet genů v jedinci, tak se hodnota automaticky upraví tak, že si rodiče vymění pouze poslední gen.

Proměnná *mutationDeviance* pro nastavení maximální odchylky během mutace je v základním nastavení nastavená na -1. Také je v třídách genetických algoritmů vytvořená podmínka, že pokud je tato proměnná menší než 0, je automaticky nastavena na jednu desetinu prohledávaného prostoru.

4.5.1. Základní verze – jednobodové křížení

Princip jednobodového křížení je popsán na Obrázek 1 Jednobodové křížení. Pravděpodobnost křížení je v základním nastavení rovna 90% a pravděpodobnost mutace je rovna 10%.

4.5.1.1. Třídní diagram

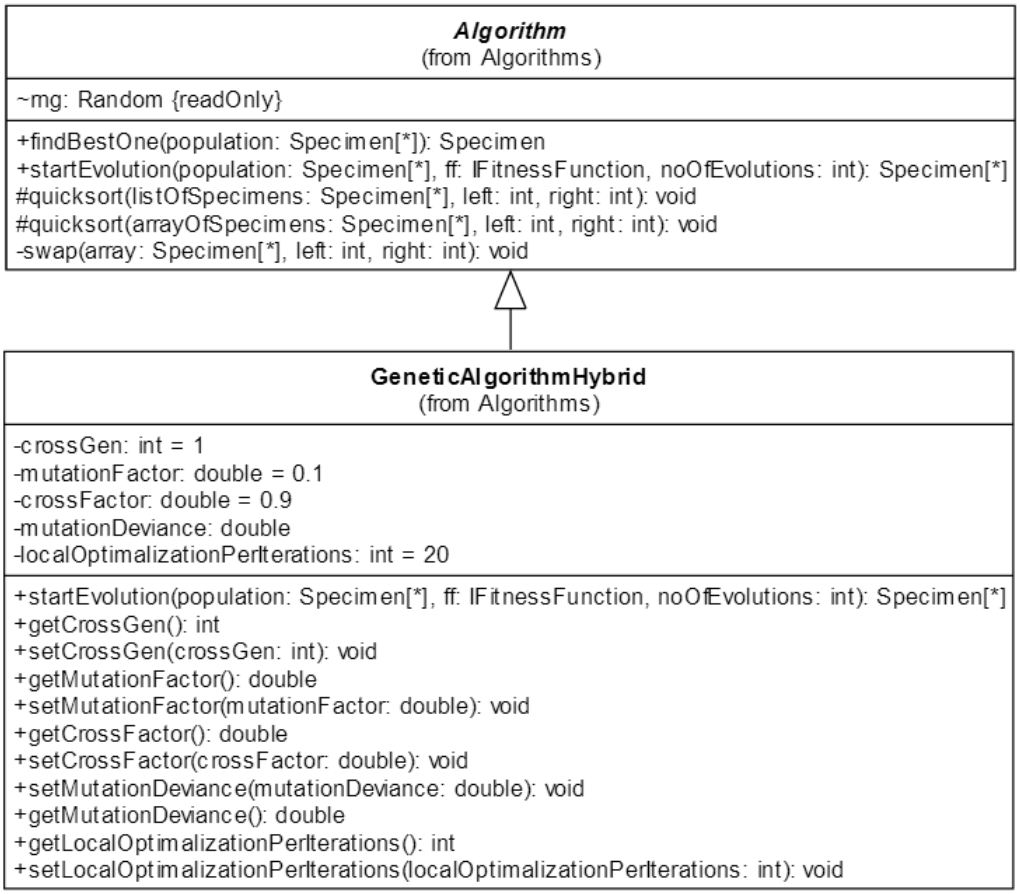


Obrázek 37 Třídní diagram - GeneticAlgorithm

4.5.2. Hybridní

Verze genetického algoritmu, který pracuje stejně jako základní verze s tím rozdílem, že obsahuje proměnnou *localOptimalizationPerIterations* v základu nastavenou na 20 iterací, která určuje, po kolika evolucích se pomocí horolezeckého algoritmu lokálně zoptimalizují všichni jedinci. S takto optimalizovanými jedinci algoritmus dál pokračuje v základním genetickém algoritmu.

4.5.2.1. Třídní diagram



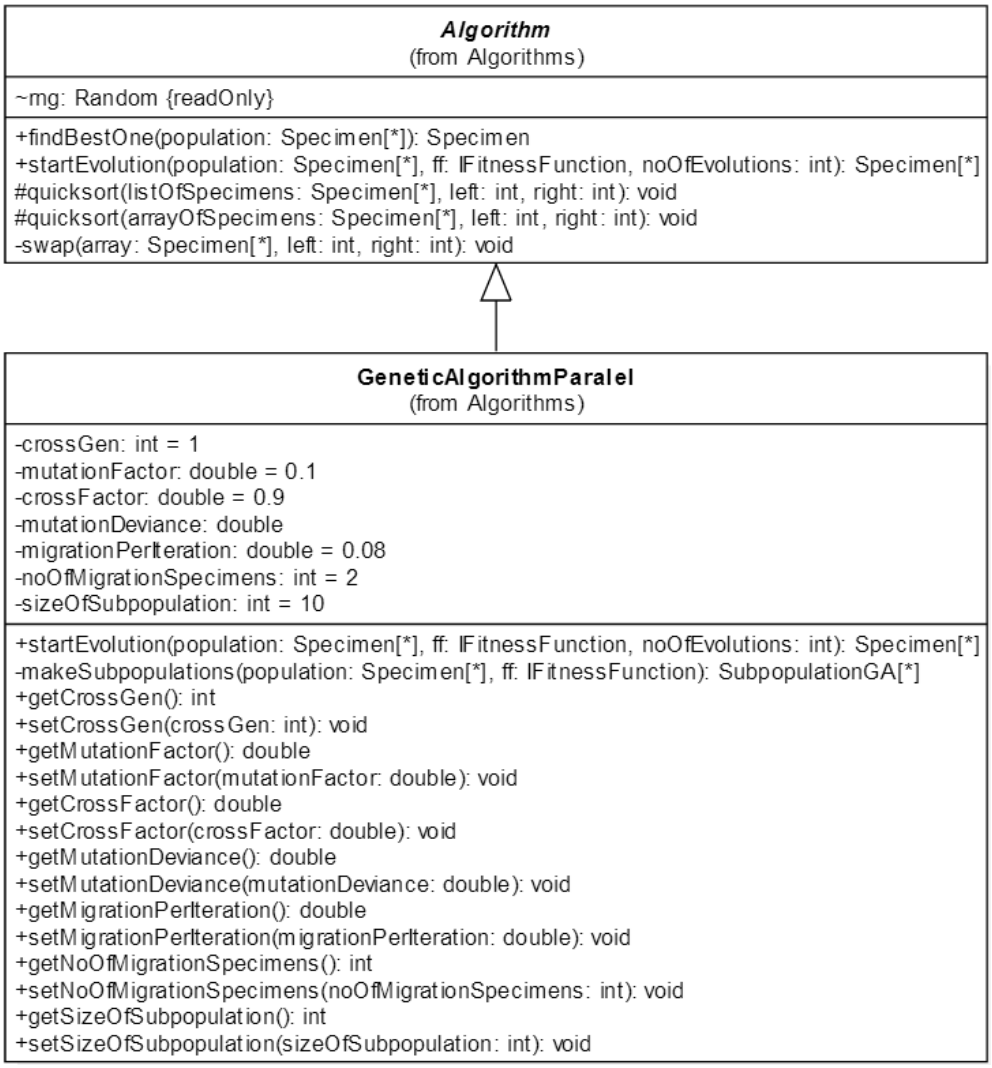
Obrázek 38 Třídní diagram - GeneticAlgorithmHybrid

4.5.3. Paralelní

Paralelní genetický algoritmus obsahuje proměnnou *migrationPerIteration*, nastavenou na hodnotu 0,08. Kód s ní dále pracuje jako s procentuální hodnotou a udává, jak často budou jedinci migrovat z jedné populace do druhé (Příklad: Při sto iteracích budou jedinci migrovat každou osmou iteraci).

Proměnná *noOfMigrationSpeciments* určuje, kolik náhodně vybraných jedinců bude migrovat do jiné, náhodně vybrané podpopulace pokaždé, když je vyvolaná migrací na základě proměnné *migrationPerIteration*.

4.5.3.1. Třídní diagram

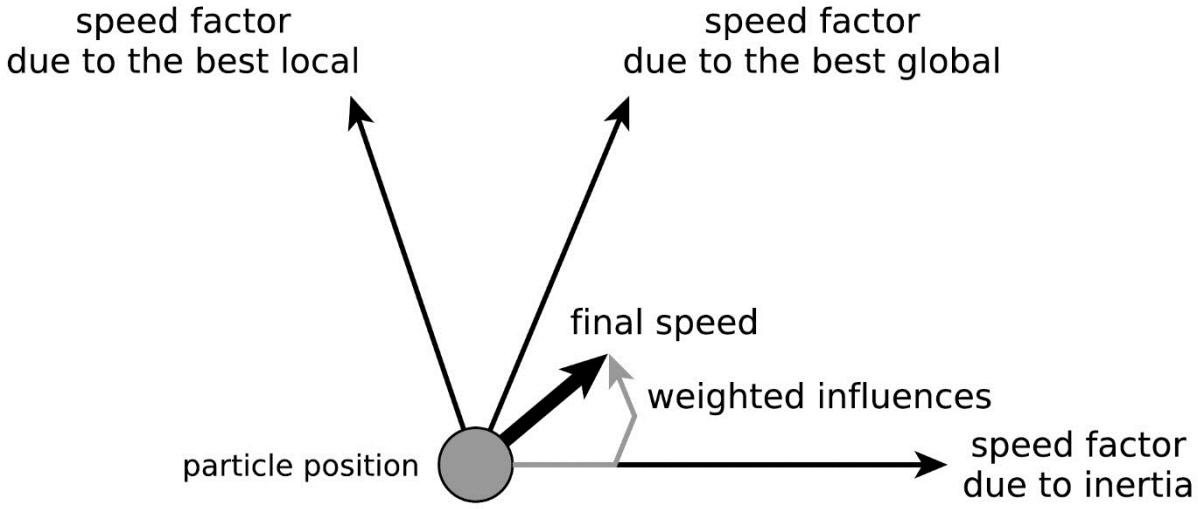


Obrázek 39 Třídní diagram – GeneticAlgorithmParalel

4.6. Rojení částic

Základní myšlenkou optimalizace pomocí roje částic se zabývá práce „*Empirical study of particle swarm optimization"* [36]. Algoritmus roje částic bývá také označován jako PSO (z anglického „Particle swarm optimalization"). Tyto algoritmy v podstatě negenerují nové potomky, jak tomu bývá u evolučních algoritmů zvykem, ale pouze pohybují již existujícími částicemi po prohledávané oblasti a hledají pozici s nejlepší vhodností. Pro tento pohyb je potřeba určit rychlost (anglicky velocity) částice, která určuje kterým směrem a jak daleko se částice v rámci jednoho migračního kola (migrační kolo zde nahrazuje pojem evoluce) posune. Na tuto rychlost mívá zpravidla vliv, kromě rychlosti z minulého migračního kola, ještě informace o doposud nejvhodnější dosažené pozici aktuální částice, doposud nejlepší dosažené vhodnosti v rámci celé populace a dosud nejlepší dosažené vhodnosti souseda. Všechny, nebo jen některé tyto informace (v závislosti na verzi PSO) mají vliv na rychlost aktuální částice a v podstatě tuto částici k sobě přitahují. Základní rovnice PSO obohacená o váhový vektor ω je definována jako (14). ω má za úkol zpomalovat rychlost částic s postupem migrace, jelikož se předpokládá, že na začátku migrace částice naleznou optimální oblast, kde by se

měl vyskytovat globální extrém a díky zpomalení částice tuto oblast podrobněji prohledají. Funkčnost zobrazuje Obrázek 40 Vliv okolí na rychlost a pozici částice [50].



Obrázek 40 Vliv okolí na rychlost a pozici částice

V knihovně je implementováno pět verzí tohoto algoritmu: základní verze s váhovým vektorem [36], verze se sousedstvím [37], „Cellular“ PSO [38], „ComprehensiveLearning“ PSO [39] a „FitnessDistanceRatioPSO“ PSO [40].

Jelikož částice vyžadují více vlastností než jedinec, jsou na začátku algoritmu jedinci převedeni privátní funkcí *makeSwarm* na třídu Particle, popsanou v kapitole 2.2. Na konci algoritmu jsou jedinci převedení zpět do třídy Specimen – kapitola 2.1.

4.6.1. Základní verze

Jedna ze základních verzí optimalizace rojem částic, která využívá krom učících faktorů ještě váhový vektor ω, ten má vliv na samotnou rychlost částice a umožnuje její zpomalování [36]. Vektor ω se upravuje před začátkem každé migrace dle rovnice (13).

$$\omega_i = ((\omega_s - \omega_s) * i) \,/\, i_{max} \quad (13)$$

ω_i… výsledná hodnota váhového faktoru pro aktuální$migraci$
ω_s… Počáteční hodnota váhového faktoru
ω_k… výsledná hodnota váhového faktoru
i… aktuální migrace
i_{max}… maximální nastavená migrace

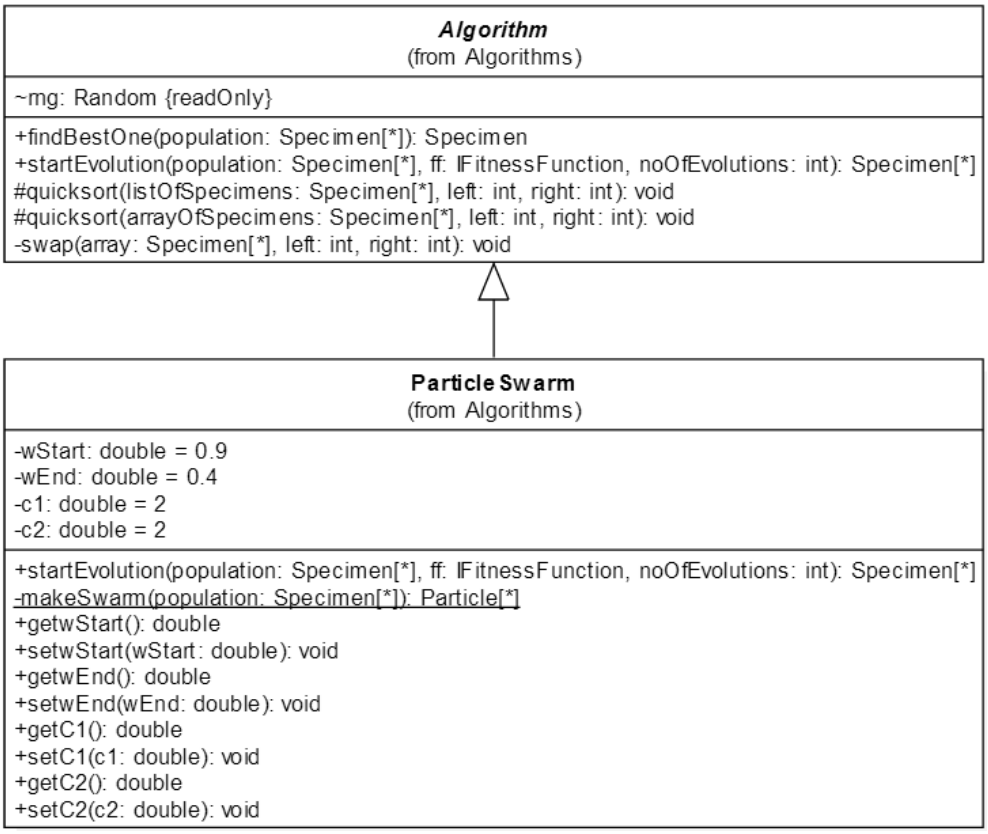
Samotná rychlost jedince se poté vypočte rovnicí (14)

$$v_{i+1} = \omega_i * v_i + c_1 * rand_1 * (pBest_x - x_i) + c_2 * rand_2 * (gBest_i - x_i) \quad (14)$$

v_{i+1}… rychlost jedince pro další migraci
ω_i… váhový vektor
ω_i… aktuální rychlost jedince
c_1… první učící faktor
c_2… druhý učící faktor
x_i… aktuální jedinec
$rand_1$… náhodně vygenerované číslo $\langle 0;1)$
$rand_2$… náhodně vygenerované číslo $\langle 0;1)$

$pBest_x$… souřadnice dosud nelepší dosažené hodnoty aktuálního jedince
$gBest_i$… souřadnice aktuálně nejlepšího řešení v celé populaci

4.6.1.1. Třídní diagram



Obrázek 41 Třídní diagram - ParticleSwarm

4.6.2. Verze se sousedstvím

Existuje více topologických verzí se sousedstvím, o kterých se dočteme něco v práci „*Particle swarm optimisation with spatial particle extension*“ [47]. V knihovně je implementována verze s kruhovou topologií kvůli jejímu pomalému toku informací o globálně nejvhodnějším jedinci. Tato informace v extrémním případě, kdy se má informace projevit na druhé straně kruhu a každá částice má dva sousedy, trvá $\frac{velikostPopulace}{2}$ kroků [47].

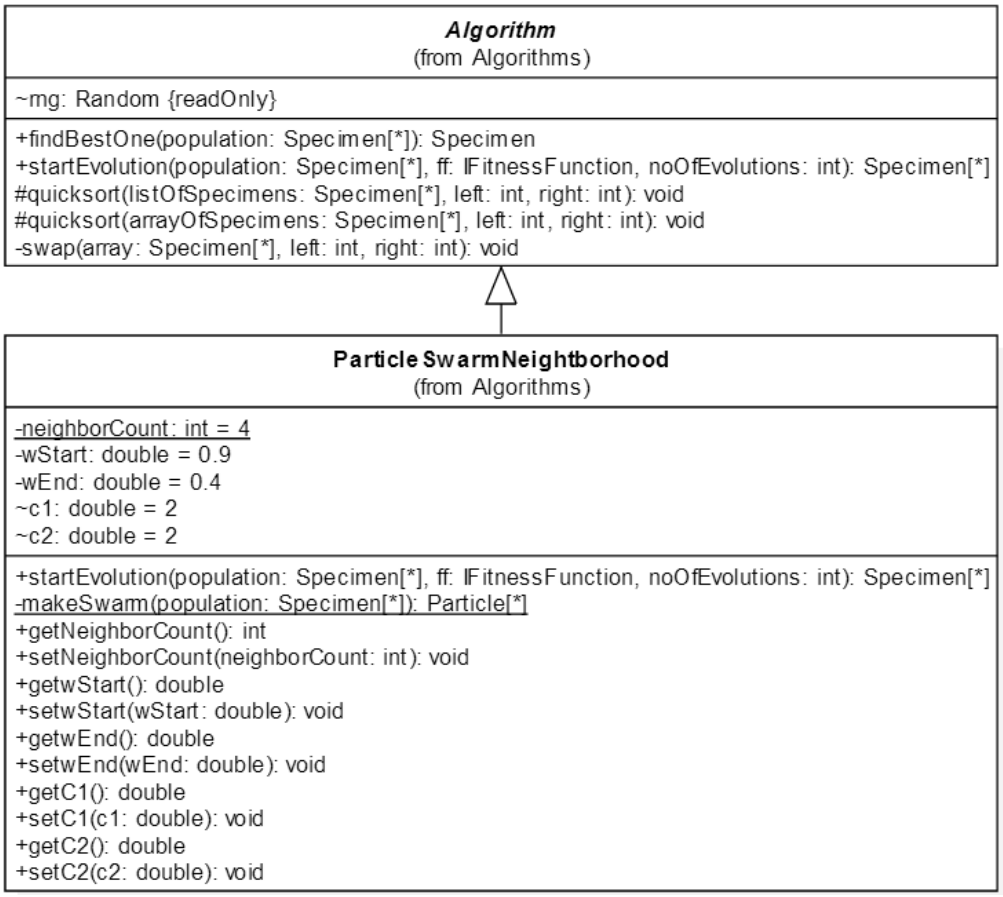
Nastavení počtu sousedů v této verzi se provádí proměnnou *neighborCount*, defaultně nastavenou na hodnotu 4. Při manuálním nastavení počtu sousedů je dobré zadávat sudé číslo. V opačném případě bude počet sousedů roven nejbližšímu většímu sudému číslu. Určení sousedství je na základě indexu v poli jedinců.

Pseudokód nastavení sousedů

for $(j = 0; j < \frac{\text{NeighborCount}}{2}; j++)$
    x_i.PridejSouseda(x_{i+x});
for $(j = 0; j < \frac{\text{NeighborCount}}{2}; j++)$
    x_i.PridejSouseda(x_{i-x});

x_i … jedinec na indexu i
neighborCount … parametr určující počet sousedů jedince

4.6.2.1. Třídní diagram

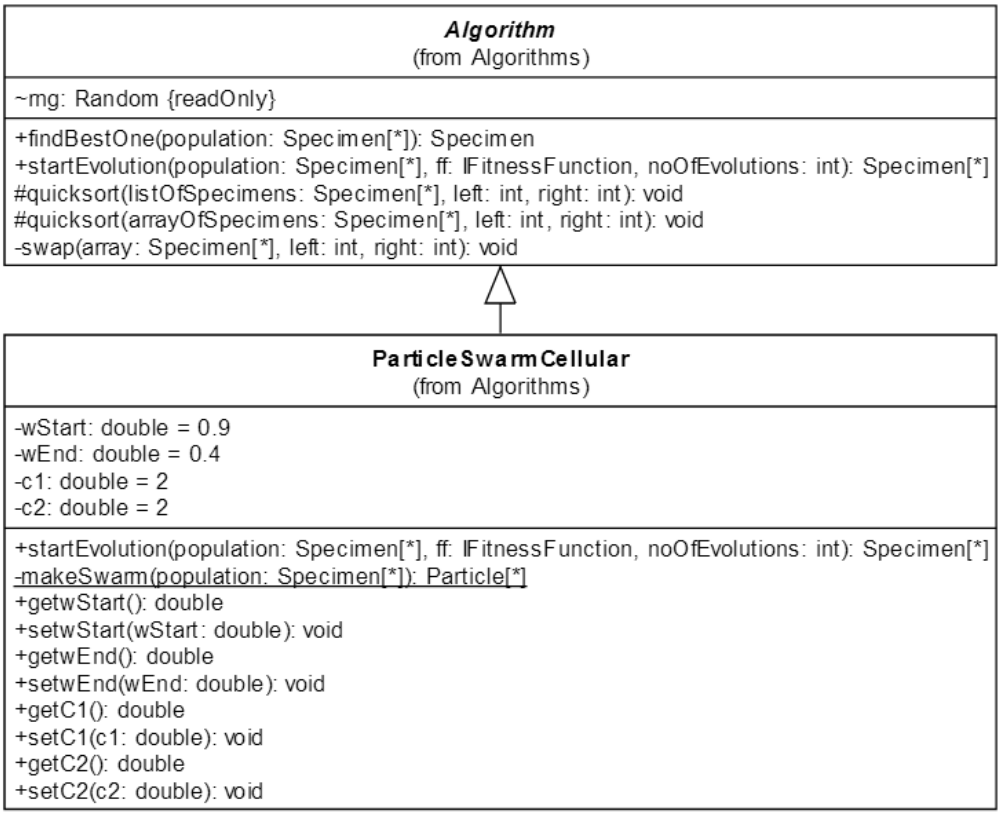


Obrázek 42 Třídní diagram - ParticleSwarmNeighborhood

4.6.3. Cellular PSO

Celulární verze je založena na sloučení základu algoritmu PSO a algoritmu celulárních automatů. Tyto dva algoritmy totiž mají jeden základní společný prvek, a to výměnu informací mezi jedinci. Každá částice je tedy vnímána jako celulární buňka a komunikuje pouze se svojí nejbližší sousedící částicí/buňkou. V rovnici PSO je globální nejlepší dosažené řešení myšleno právě nejlepším řešením nejbližšího sousedícího jedince [38]. Změna globálně nejlepší částice za sousední částici zpomaluje rychlý přesun všech částic směrem k doposud nejlepšímu nalezenému řešení, díky pomalejší distribuci informace, kde se toto řešení nachází. Částice tedy nemají tendenci shlukovat se rovnou kolem nejlepší z nich, ale postupují pomalu jedna k druhé a pečlivěji prohledávají oblasti [38].

4.6.3.1. Třídní diagram



Obrázek 43 Třídní diagram - ParticleSwarmCellular

4.6.4. ComprehensiveLearning PSO

Typ PSO, ve kterém je vylepšena konvergence algoritmu pomocí odstranění závislosti na globálně nejlepší částici v celé populaci [39]. Dále je modifikována část vzorce, kdy je částice přitahována ke svému nejlepšímu nalezenému řešení a to tak, že se před generací nové rychlosti částice náhodně vyberou dvě jiné částice. Z nich se vybere ta, která má lepší hodnotu vhodnosti. Tato náhodně vybraná částice má šanci ovlivnit výpočet rychlosti aktuální částice svým, doposud nejlepším nalezeným řešením. Šance na ovlivnění je P_c a ta je vypočtena podle následující rovnice (15).

$$P_c = 0{,}05 + 0{,}45 * \frac{\exp\left(\frac{10*j}{PS-1}\right)-1}{\exp(10)-1} \tag{15}$$

P_c… pravděpodobnost učení se od jiné částice
j… index aktuálního genu číslován od nuly
PS… Celkový počet částic v populaci

Následně se postupuje dle vzorce (16)

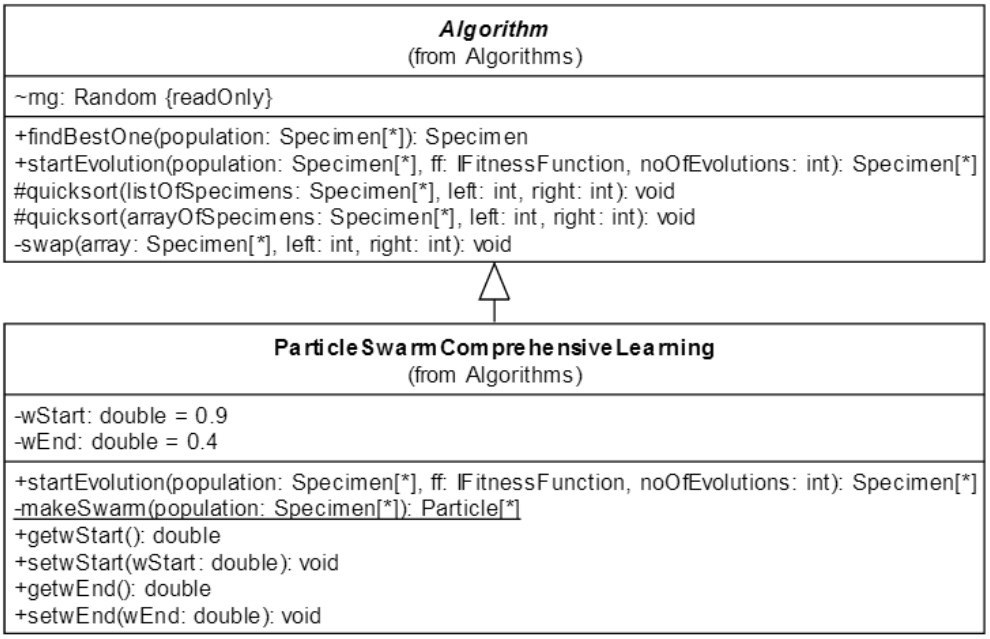
$$\begin{cases} if\ rand_1 < P_c,\ then\ v_{i+1} = \omega * v_i + c_1 * rand_2 * (pBest_r - x_i) \\ if\ rand_1 \geq P_c,\ then\ v_{i+1} = \omega * v_i + c_1 * rand_2 * (pBest_x - x_i) \end{cases} \tag{16}$$

P_c… pravděpodobnost učení se od jiné částice
v_{i+1}… nová rychlost částice
v_i… aktuální rychlost částice
ω… váhový vektor
c_1… učící faktor

x_i… aktuálního částice
$rand_1$… první náhodné číslo
$rand_2$… druhé náhodné číslo
$pBest_r$… nejlepší nalezené řešení lepšího ze dvou náhodně vybraných částic
$pBest_x$… nejlepší nalezené řešení aktuální částice

Nakonec musí být zajištěno, že aktuální částice musí být ovlivněna jinou částicí alespoň v jednom genu [39].

4.6.4.1. Třídní diagram



Obrázek 44 Třídní diagram – ParticleSwarmComprehensiveLearning

4.6.5. FitnessDistanceRatio PSO

Verze PSO, která přidala do základní rovnice pro ovlivnění rychlosti částice ještě jednu část, a to nejlepší nalezené řešení sousední částice, která je vůči aktuální částici v nejlepším poměru vhodnost/vzdálenost. Tento poměr získáme rovnicí (17)[40].

$$FDR = \frac{x_{2fit} - x_{1fit}}{x_{2i} - x_{1i}} \quad (17)$$

FDR… poměr vhodnost / vzdálenost
x_{1fit}… vhodnost aktuální částice
x_{2fit}… vhodnost porovnávané částice
x_{1i}… hodnota genu aktuální částice
x_{2i}… hodnota genu porovnávané částice

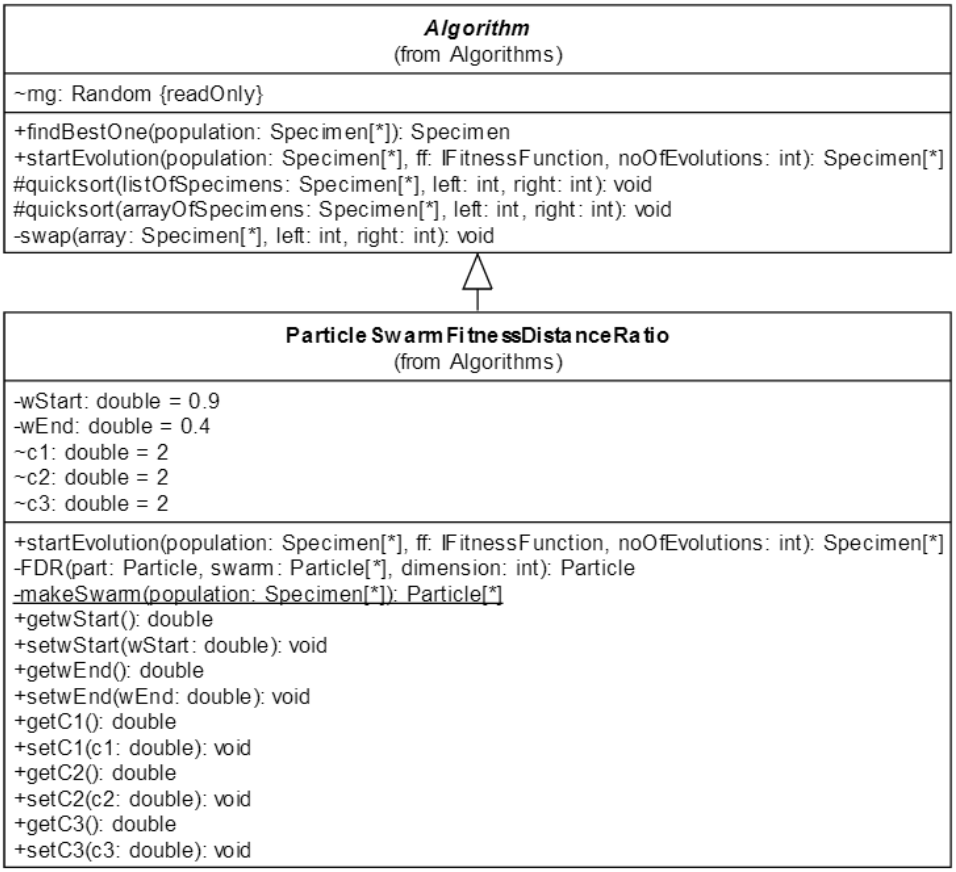
Pomocí FDR poměru vybereme nejvhodnější částici a zařadíme ji do rovnice (18) pro výpočet nové rychlosti částice [40].

$$v_{i+1} = \omega * v_i + c_1 * rand_1 * (pBest_i - x_i) + c_2 * rand_2 * (gBest_i - x_i) + c_3 * rand_3 * (fdrBest_i - x_i) \quad (18)$$

v_{i+1}… nová rychlost částice

v_i… aktuální rychlost částice
ω… váhový vektor
c_1… první učící faktor
c_2… druhý učící faktor
c_3… třetí učící faktor
x_i… aktuálního částice
$rand_1$… první náhodné číslo
$rand_2$… druhé náhodné číslo
$rand_3$… třetí náhodné číslo
$gBest_i$… nejlepší nalezené řešení v aktuální populaci
$pBest_i$… nejlepší nalezené řešení aktuálního částice
$fdrBest_i$… nejlepší nalezené řešení částice s nejvhodnějším poměrem vhodnost/vzdálenost

4.6.5.1. Třídní diagram



Obrázek 45 Třídní diagram - ParticleSwarmFitnessDistanceRatio

4.7. Samo organizující se migrační algoritmus - SOMA

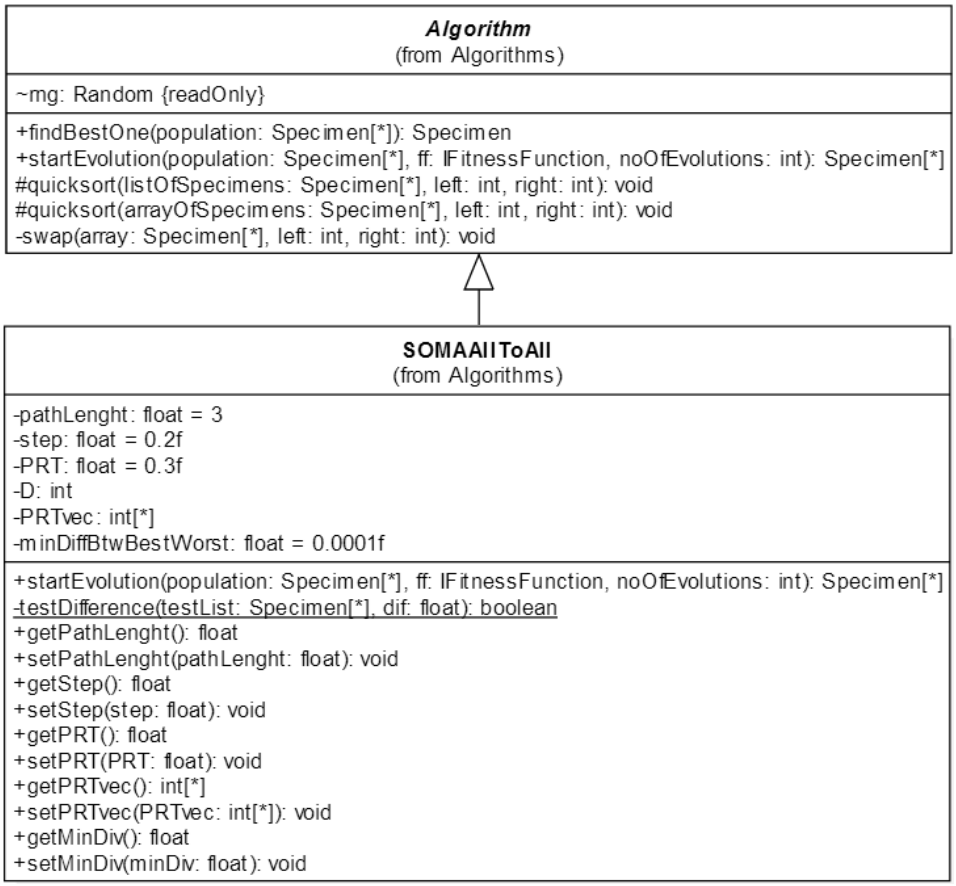
Více o algoritmu nalezneme v pracích I. Zelinky [2,25]. V knihovně jsou tři typy migrací, *„All to One“, „All to All“* a *„All to One Rand“*. Každá migrace je navíc v adaptivní verzi, která se liší v tom, že posun každého jedince je okamžitě zaznamenáván do populace v průběhu migračního kola, ne až

po ukončení migračního kola, jak je tomu ve standardních verzích. To může mít za příčinu například několikanásobnou změnu nejlepšího jedince, ke kterému ostatní jedinci migrují, během jednoho migračního kola, ve variaci „*All To One*“ 4.7.1 [2].

4.7.1. All To One

Jednoduchá verze SOMA algoritmu, kde všichni jedinci zkoumají cestu k nejlepšímu jedinci v rámci migračního kola. Tento jedinec, ke kterému ostatní migrují, je označován jako leader, ke kterému se ostatní pohybují pomocí skoku, jejichž délka je daná parametrem *step*. Po každém skoku si jedinec zapamatuje novou hodnotu vhodnosti. Nakonec, po dokončení všech skoků vůči leaderovi, se aktuální jedinec vrátí na místo, kde měl nejlepší vhodnost [2].

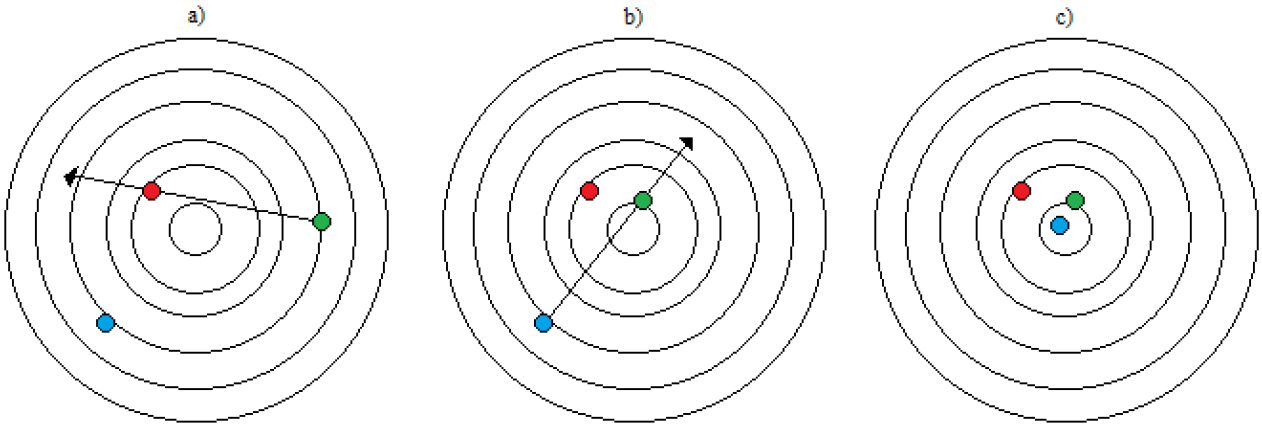
4.7.1.1. Třídní diagram



Obrázek 46 Třídní diagram –SOMAAllToOne

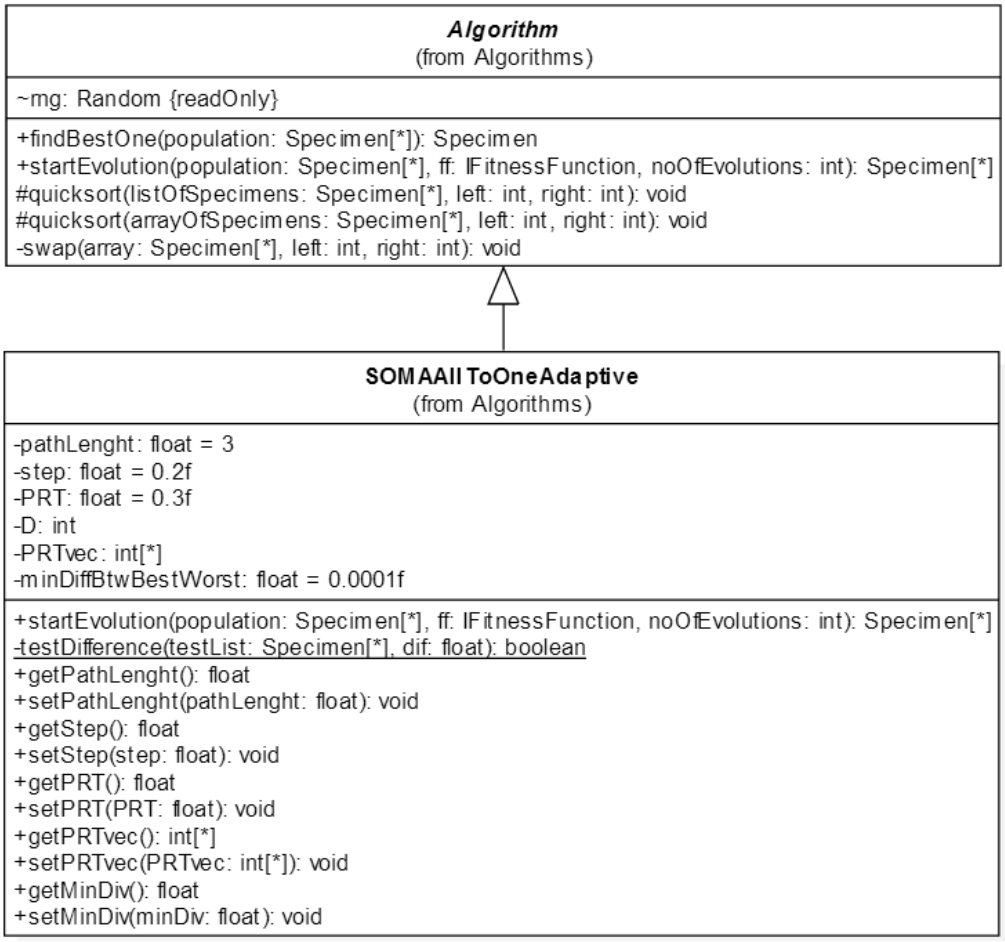
4.7.2. All To One - Adaptive

Implementována *„All To One“* verze algoritmu SOMA s modifikací adaptive. Jak tato modifikace funguje lze vidět na Obrázek 47 SOMA adaptivní migrace a) před migrací b) první krok migrace c) druhý krok migrace



Obrázek 47 SOMA adaptivní migrace a) před migrací b) první krok migrace c) druhý krok migrace

4.7.2.1. Třídní diagram



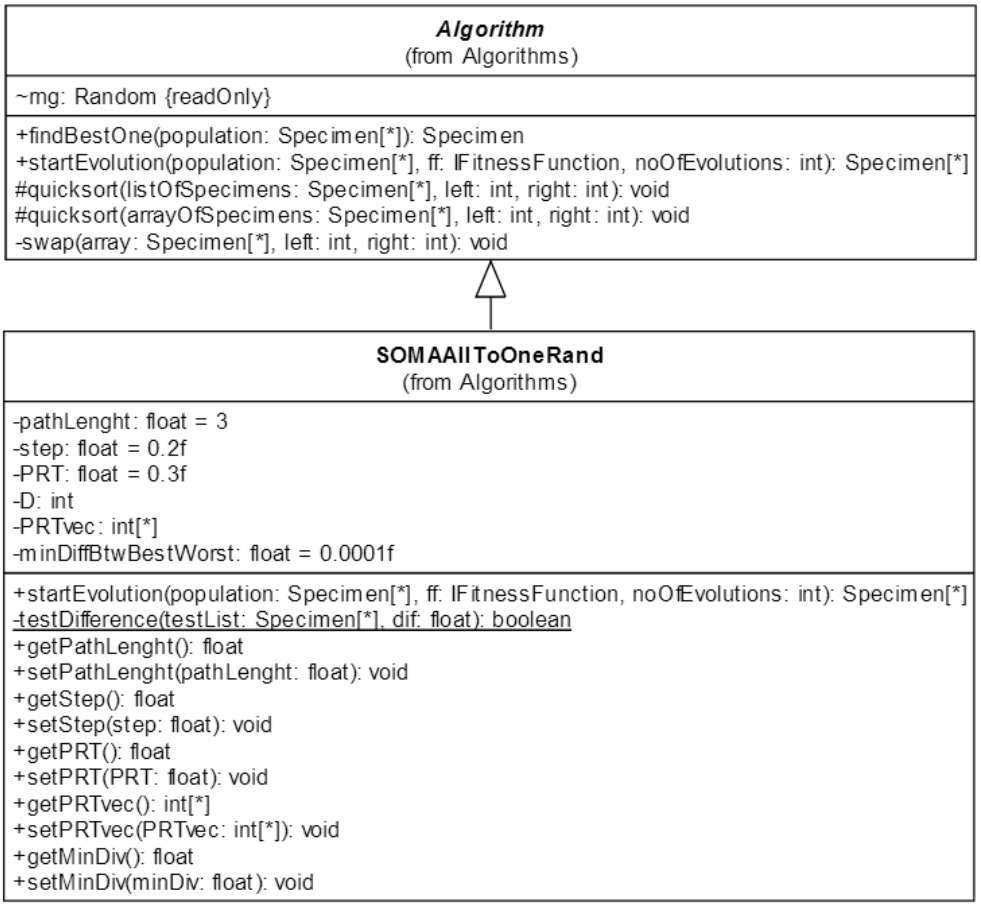
Obrázek 48 Třídní diagram - SOMAAllToOne– Adaptive

4.7.3. All To One Rand

SOMA modifikace, která funguje stejně jako verze „*All To One*“, akorát za leadera není určen jedinec s nejlepší vhodností v populaci, ale leader je určen náhodně z celé populace a pro každého jedince

znova. Tento jedinec pak migruje k tomuto leaderovi, který klidně může mít nejhorší vhodnost v populaci. V pesimistické verzi tedy všichni mohou migrovat k nejhoršímu jedinci v populaci [2].

4.7.3.1. Třídní diagram

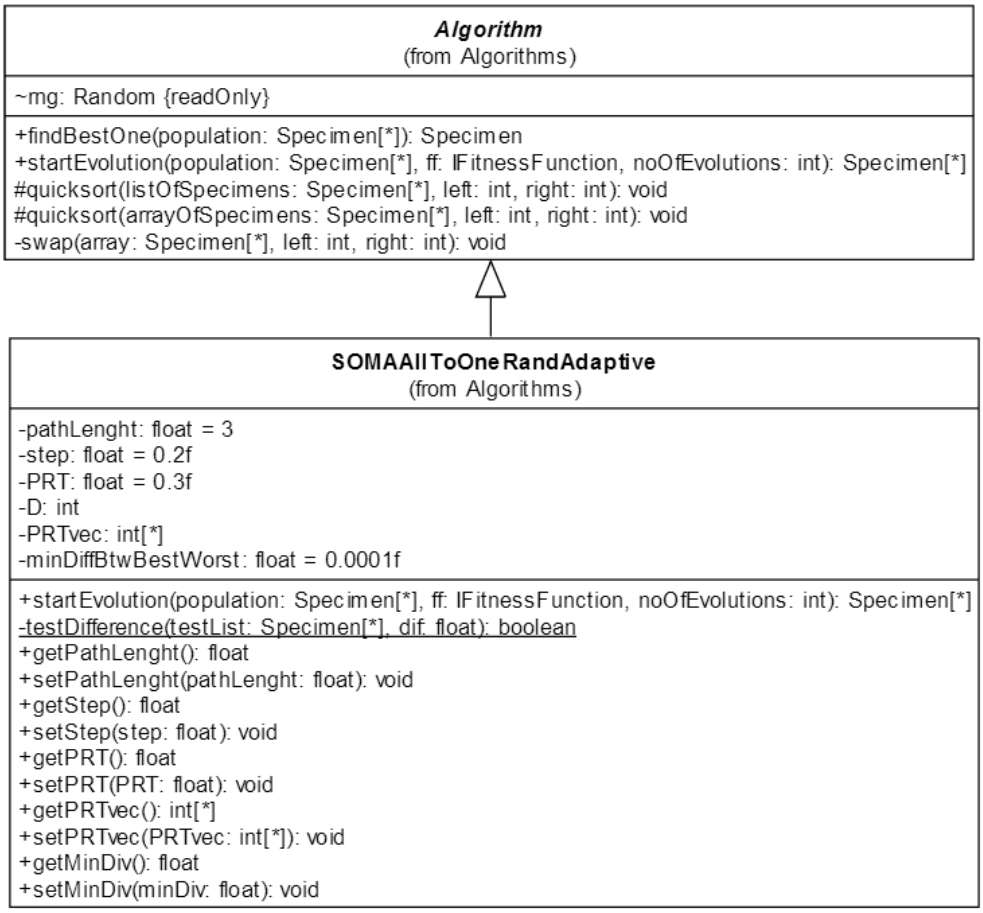


Obrázek 49 Třídní diagram – SOMAAllToOneRand

4.7.1. All To One Rand – Adaptive

Adaptivní verze SOMA „*All To One Rand*“ zaznamenává přesun jedince v populaci okamžitě po jeho pohybu, ne až na konci migračního kola. Ostatní tedy mohou migrovat k takto posunutému jedinci během stejného migračního kola [2].

4.7.1.1. Třídní diagram

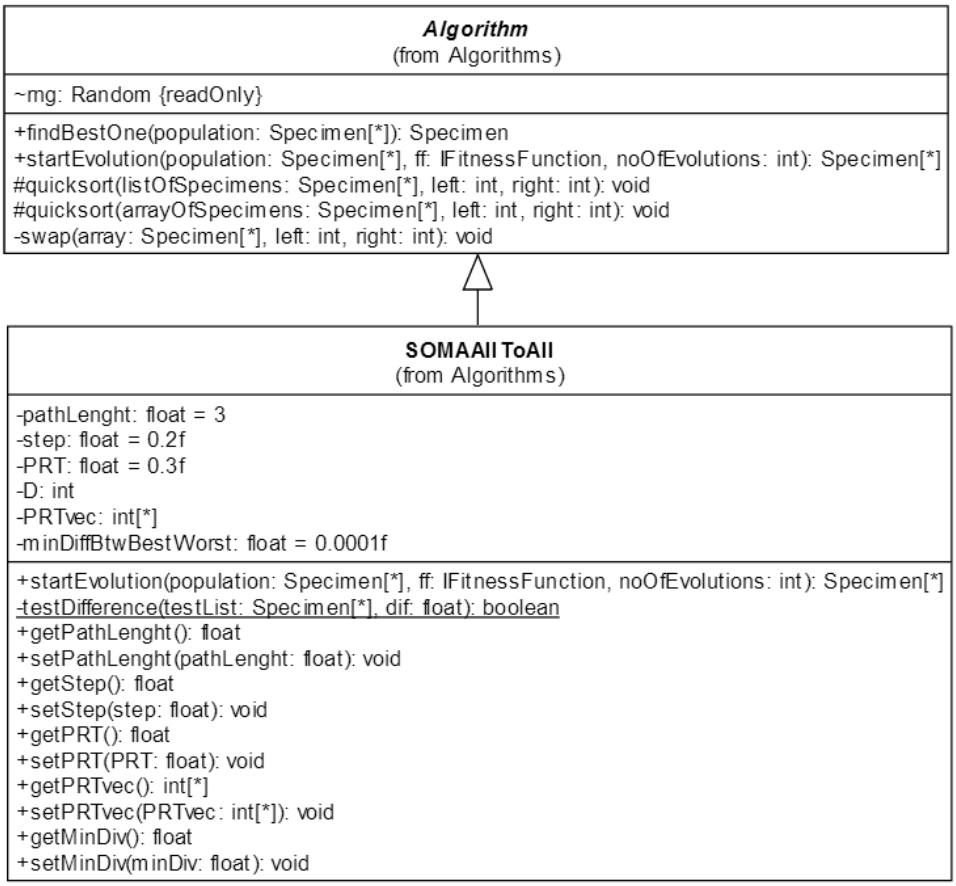


Obrázek 50 Třídní diagram - SOMAAllToOneRand – Adaptive

4.7.2. All To All

Každý jedinec zkoumá cesty stejně jako ve verzi *„All To One“*, ale tyto cesty zkoumá vůči všem ostatním jedincům v rámci migrace, neexistuje zde tedy žádný leader. Poté, co každý jedinec prozkoumá všechny cesty vůči ostatním, jsou všichni posunuti na pozici, kde jedinec nalezl nejlepší extrém, na svých *popSize-1* zkoumaných cestách. Tato verze je výpočetně náročná, jelikož jedinec během migrace zkoumá mnohem větší prostor možných řešení, za cenu větší šance nalézt globální extrém [2]. Algoritmus je možno zrychlit, ale také znepřesnit zvýšením hodnoty *step* a snížením hodnoty *pathLenght*.

4.7.2.1. Třídní diagram

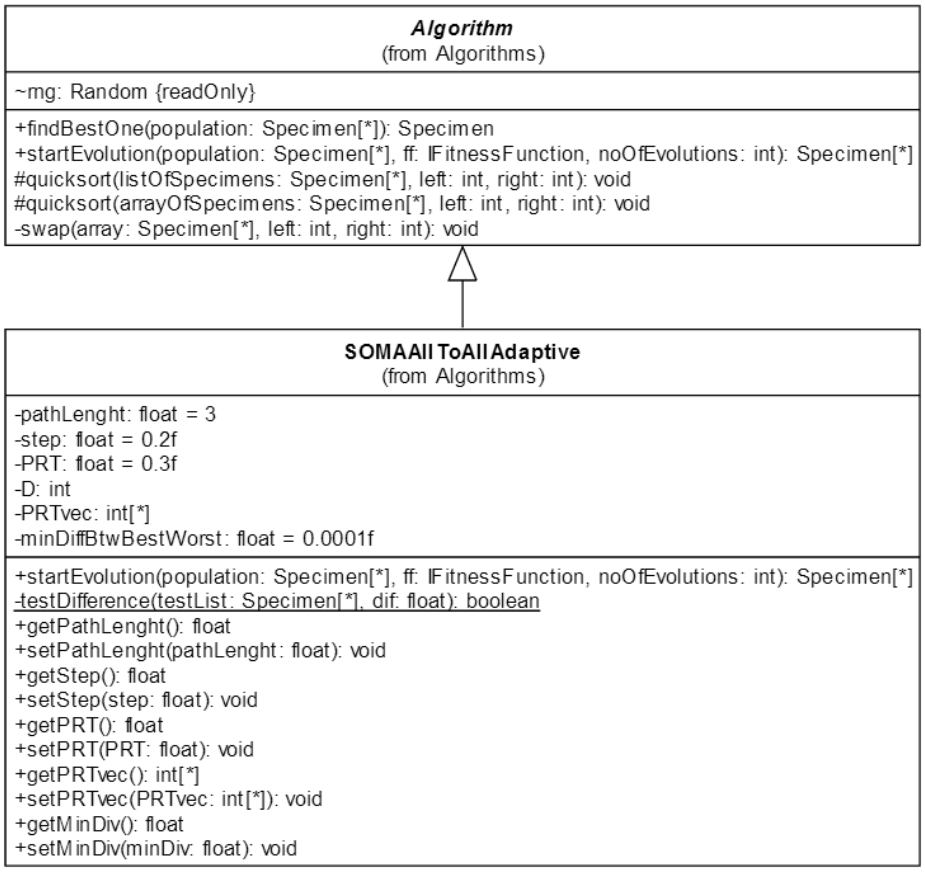


Obrázek 51 Třídní diagram – SOMAAllToAll

4.7.3. All To All – Adaptive

Adaptivní verze SOMA „All To All“ zaznamenává pohyb jedince do populace během migračního kola, ne až po jeho dokončení, takže každý jedinec, který nezačíná migraci, prozkoumává cesty k nově nalezeným extrémům jedinců, kteří prohledávali možná řešení před ním.

4.7.3.1. Třídní diagram



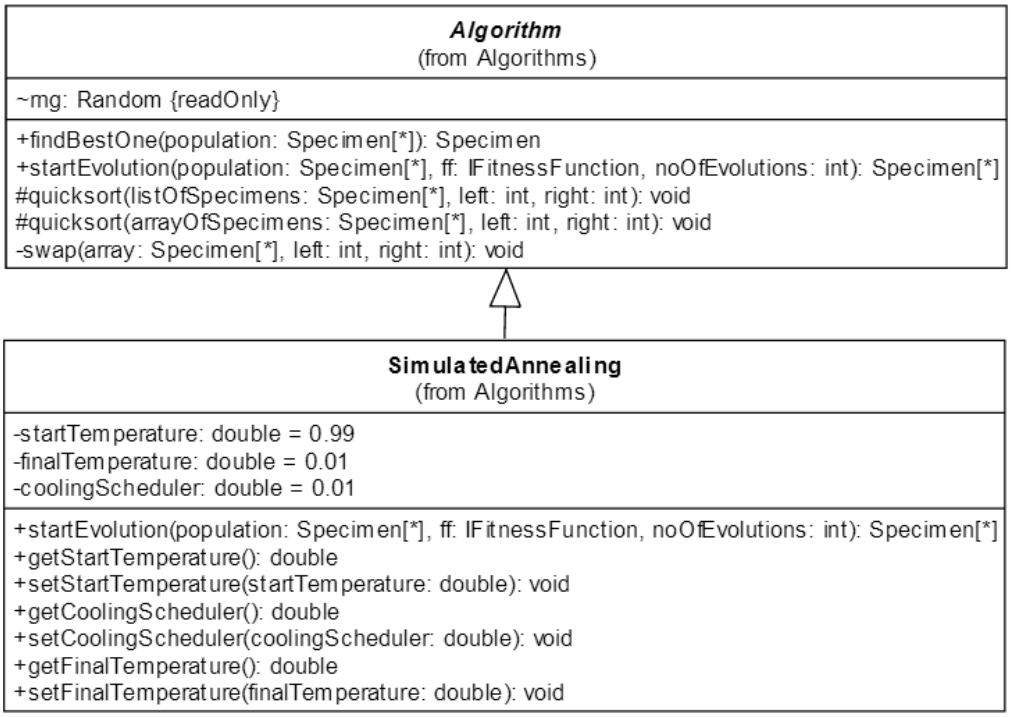
Obrázek 52 Třídní diagram - SOMAAllToAll – Adaptive

4.8. Simulované žíhání

Problematikou simulovaného žíhání se zabývá kniha *Simulated annealing* [41]. Simulované žíhání patří mezi optimalizaci pro lokální extrémy (stejně jako například horolezecký algoritmus), ale jelikož navíc připouští zhoršení vhodnosti jedince, je schopen, alespoň ze začátku, překonat lokální extrémy. Inspirací k tomuto procesu je fyzikální proces žíhání tuhého tělesa. Stejně jako v metalurgii, je kov zahřát na vysokou teplotu, a pak velmi pomalu ochlazován, i zde je počáteční teplota a plán chlazení, pomocí kterého se dopočítává aktuální teplota pro každou generaci. Evoluce probíhá tak dlouho, dokud nedosáhneme konečné teploty, která značí konec evoluce. Schopnost jedince přijmout řešení s horší vhodností je větší při vyšších teplotách, tedy v prvních generacích evoluce, při nižších teplotách se optimalizace chová spíše jako horolezecký algoritmus [41].

Aby evoluce mohla být ukončena pomocí chladícího plánu, je dobré volat funkci s parametrem *noOfEvolution* nastavenou na maximální hodnotu třídy Integer, konkrétně 2^{31}-1 [6].

4.8.1. Třídní diagram

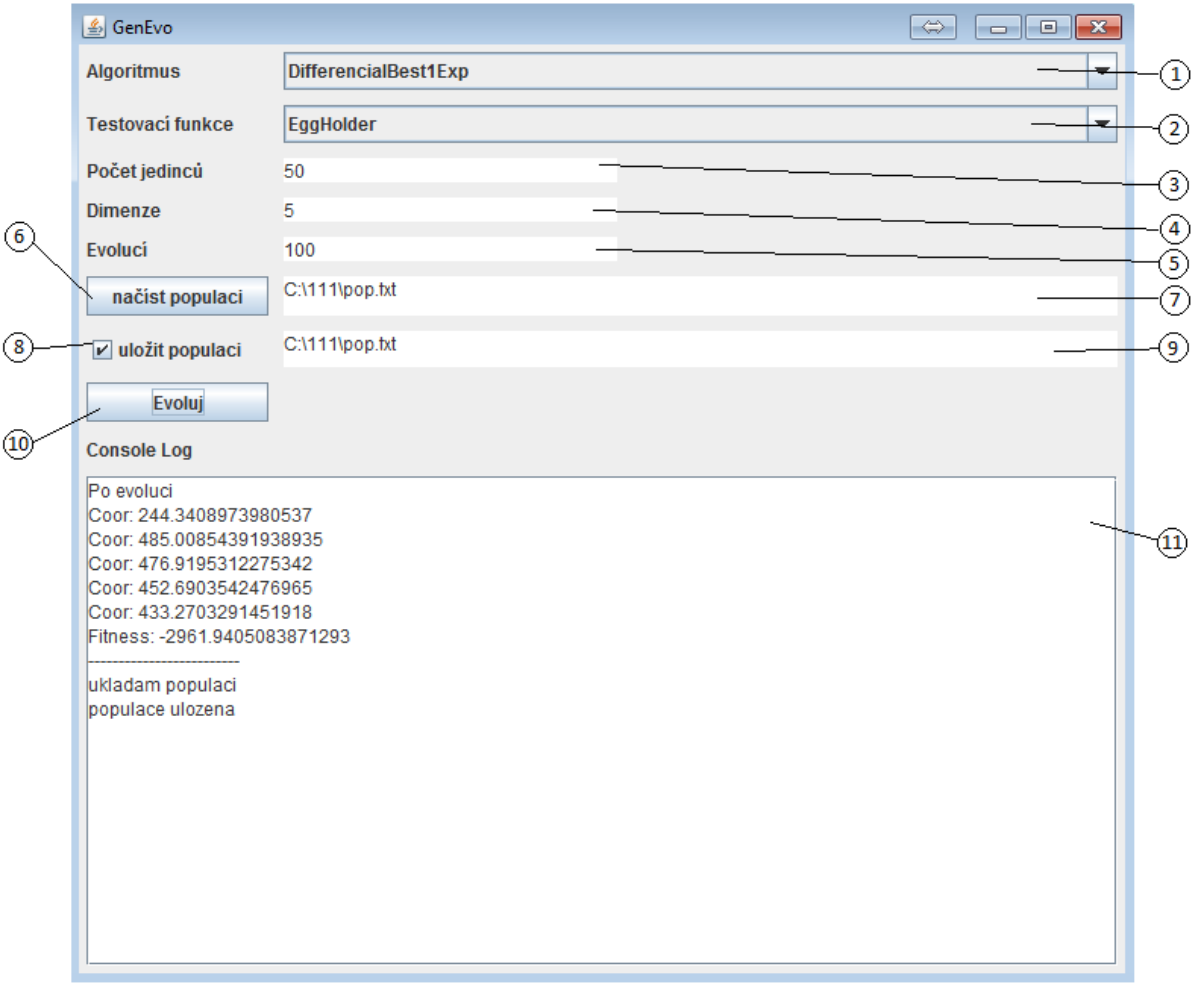


Obrázek 53 Třídní diagram - SimulatedAnnealing

5. Grafické rozhraní

Grafické rozhraní projektu slouží pouze pro demonstraci základních funkcí knihovny. Není v něm možno nastavovat parametry tříd.

5.1. Popis rozhraní



Obrázek 54Grafické rozhraní programu pro demonstraci algoritmů

1. Seznam dostupných evolučních algoritmů.
2. Seznam dostupných testovacích funkcí.
3. Nastavení počtu nově generovaných jedinců.
4. Nastavení dimenze prohledávaného problému.
5. Nastavení počtu evolucí jedinců, pro hledání globálního optima.
6. Načte jedince a jejich vlastnosti ze souboru.
7. Určuje cestu k souboru, ze kterého se mají načítat jedinci.
8. Zaškrtávací pole definující, zdali se má populace po evoluci uložit do souboru, aby ji bylo možno později použít pomocí tlačítka pro načtení populace.
9. Určuje cestu k souboru, do kterého se má uložit populace po evoluci
10. Tlačítko zahajující evoluci populace
11. Log programu, sloužící pro předávání informací.

5.1.1. Vlastnosti rozhraní

Seznamy s evolučními algoritmy a testovací funkce jsou načítány z knihovny GenEvoLib.jar pomocí třídy ZipEntry, která dokáže načíst .jar soubory a procházet jejich obsah [7]. Proto musí být soubor s knihovnou ve složce lib, která je v podadresáři složky se spouštěcím souborem.

Při nastavení počtů dimenzí na dvě, se po evoluci zobrazí tři okna s grafy, na kterých jde vidět proces evoluce. První okno zobrazuje počáteční rozmístění jedinců, na druhém okně lze vidět, jak vypadala populace v polovině evoluce a na posledním okně se zobrazuje populace po dokončení evoluce. Grafy jsou vytvořené pomocí Java OpenGL (JOGL), konkrétně open source knihovny Jzy3d [8], takže grafy lze pomocí myši natáčet a roztahovat.

Pro zobrazení oken s grafy je potřeba mít nainstalované JRE7, do této složky je poté potřeba importovat knihovny .jar a .dll se kterými JOGL pracuje. Pro jednoduchost jsou vytvořeny a přiloženy dva zipy, které stačí rozbalit do své složky s JRE. První zip obsahuje knihovny pro 32-bitovou Java, druhý zip obsahuje knihovny pro 64-bitovou verzi Java.

Po zmáčknutí tlačítka pro načtení populace se pro následující evoluci použije načtená populace, zároveň se textová pole počet jedinců a dimenze znepřístupní pro editaci a nastaví se do nich hodnoty načtené populace. Znepřístupnění editace trvá do zmáčknutí tlačítka Evoluj.

Uložená populace se do souboru ukládá ve formátu [x_1_x_2_ … _x_i_]:fit kde x_i je vlastnost jedince a fit je vhodnost jedince.
Dvoučlenná populace o třech dimenzích tedy může vypadat následovně:
-511.90383632615277_-341.92737706654236_-170.8554011358157_:-1430.2925575771383
-476.87740537078196_121.90427625575437_-380.0766128100575_:-927.983899616635

V logu programu se zobrazují následující informace

- Vlastnost a vhodnost nejlepšího jedince po evoluci
- Ukládání a načtení populace
- Jakákoliv chyba, která vznikne při běhu programu

6. Závěr

Práce přinesla novou knihovnu optimalizačních algoritmů v jazyce Java, která je snadno rozšiřitelná, připravená pro otestování nově implementovaných optimalizačních i testovacích funkcí a snadno importovatelná do jiného projektu.

Knihovna navíc obsahuje převratné množství algoritmů v jednom balíčku, z toho několik nových algoritmů, které v jiných knihovnách doposud nenalezneme. Například algoritmus HSDE nebo IMDE, jelikož byli publikovány v roce 2013. Algoritmy SOMA v profesionálních knihovnách nenalezneme vůbec. V této práci jsou tři s alternativní, adaptivní, verzí. Všechny algoritmy rojení částic, které jsou v této knihovně, jsou takto pohromadě také raritou.

Také zde nalezneme několik testovacích funkcí, které je možno použít jako základ pro vytvoření benchmarku.

6.1. Zdroje

[1] LIANG, J. J.; QU, B. Y.; SUGANTHAN, P. N. Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization. *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*, 2013.

[2] ZELINKA, I., et al. Evoluční výpočetní techniky–principy a aplikace. BEN–technická literatura, Praha, 2008. ISBN 80-7300-218-3.

[3] PRICE, Kenneth; STORN, Rainer M.; LAMPINEN, Jouni A. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[4] Normal(Java Statistical Classes). [online]. [cit. 2015-04-06]. Dostupné z: http://www.jsc.nildram.co.uk/api/jsc/distributions/Normal.html

[5] Double (Java Platform SE7). [online]. [cit. 2015-04-06]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/lang/Double.html

[6] Integer (Java Platform SE 7). [online]. [cit. 2015-04-06]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html

[7] ZipEntry (Java Platform SE 7). [online]. [cit. 2015-04-06]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/util/zip/ZipEntry.html

[8] Jzy3D - scientific 3d plotting. [online]. [cit. 2015-04-06]. Dostupné z: http://jzy3d.org/

[9] index.php. *The Watchmaker Framework for Evolutionary Computation (evolutionary/genetic algorithms for Java).* [online]. 25.4.2015 [cit. 2015-04-25]. Dostupné z: *http://watchmaker.uncommons.org/index.php*

[10] *Jenes – Genetic Algorithms for Java.* [online]. 25.4.2015 [cit. 2015-04-25]. Dostupné z: *http://sourceforge.net/projects/jenes/*

[11] Sean Luke, Liviu Panait, Gabriel Balan, Sean Paus, Zbigniew Skolicki, Rafal Kicinger, Elena Popovici, Keith Sullivan, Joseph Harrison, Jeff Bassett, Robert Hubley, Ankur Desai, Alexander

Chircop, Jack Compton, William Haddon, Stephen Donnelly, Beenish Jamil, Joseph Zelibor, Eric Kangas, Faisal Abidi, Houston Mooers, James O\'Beirne, Khaled Ahsan Talukder, and James McDermott. . *ECJ*. [online]. 25.4.2015 [cit. 2015-04-25]. Dostupné z: http://cs.gmu.edu/~eclab/projects/ecj/

[12] BACK, Thomas; HAMMEL, Ulrich; SCHWEFEL, H.-P. Evolutionary computation: Comments on the history and current state. *Evolutionary computation, IEEE Transactions on*, 1997, 1.1: 3-17.

[13] BREMERMANN, Hans J. Optimization through evolution and recombination. Self-organizing systems, 1962, 93: 106.

[14] FRIEDBERG, Richard M. A learning machine: Part I. IBM Journal of Research and Development, 1958, 2.1: 2-13.

[15] BOX, George EP. Evolutionary operation: A method for increasing industrial productivity. Applied Statistics, 1957, 81-101.

[16] BERTIE, Andrew. JSC: Java Statistical Classes. [online]. [cit. 2015-04-29]. Dostupné z: http://www.jsc.nildram.co.uk/

[17] HOLLAND, John H. Genetic algorithms. *Scientific american*, 1992, 267.1: 66-72.

[18] ZHANG, Jingqiao; SANDERSON, Arthur C. JADE: adaptive differential evolution with optional external archive. Evolutionary Computation, IEEE Transactions on, 2009, 13.5: 945-958.

[19] QIN, A. Kai; SUGANTHAN, Ponnuthurai N. Self-adaptive differential evolution algorithm for numerical optimization. In: *Evolutionary Computation, 2005. The 2005 IEEE Congress on*. IEEE, 2005. p. 1785-1791.

[20] YANG, Zhenyu; TANG, Ke; YAO, Xin. Self-adaptive differential evolution with neighborhood search. In: *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*. IEEE, 2008. p. 1110-1116.

[21] STORN, Rainer; PRICE, Kenneth. *Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces*. Berkeley: ICSI, 1995.

[22] EIGEN, Manfred. Ingo Rechenberg Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologishen Evolution. mit einem Nachwort von Manfred Eigen, Friedrich Frommann Verlag, Struttgart-Bad Cannstatt, 1973.

[23] BÄCK, Thomas. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary* programming*, genetic algorithms*. New York: Oxford University Press, 1996, xii, 314 p.

[24] EBERHART, Russ C.; KENNEDY, James. A new optimizer using particle swarm theory. In: *Proceedings of the sixth international symposium on micro machine and human science*. 1995. p. 39-43.

[25] ZELINKA, Ivan. SOMA—self-organizing migrating algorithm. In: *New optimization techniques in engineering*. Springer Berlin Heidelberg, 2004. p. 167-217.

[26] MISHRA, Sudhanshu K. Some new test functions for global optimization and performance of repulsive particle swarm method. *Available at SSRN 926132*, 2006.

[27] MOLGA, Marcin; SMUTNICKI, Czesław. Test functions for optimization needs. *Test functions for optimization needs*, 2005.

[28] SELMAN, Bart; GOMES, Carla P. Hill-climbing Search.

[29] WANG, Yong; CAI, Zixing; ZHANG, Qingfu. Differential evolution with composite trial vector generation strategies and control parameters. *Evolutionary Computation, IEEE Transactions on*, 2011, 15.1: 55-66.

[30] BREST, Janez; MAUČEC, Mirjam Sepesy. Self-adaptive differential evolution algorithm using population size reduction and three strategies. *Soft Computing*, 2011, 15.11: 2157-2174.

[31] ZHOU, Yinzhi; LI, Xinyu; GAO, Liang. A differential evolution algorithm with intersect mutation operator. *Applied Soft Computing*, 2013, 13.1: 390-401.

[32] WANG, Lin, et al. An effective hybrid self-adapting differential evolution algorithm for the joint replenishment and location-inventory problem in a three-level supply chain. *The Scientific World Journal*, 2013, 2013.

[33] STORN, Rainer; PRICE, Kenneth. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 1997, 11.4: 341-359.

[34] DAVIS, Lawrence, et al. (ed.). *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold, 1991.

[35] MITCHELL, Melanie. *An introduction to genetic algorithms*. MIT press, 1998.

[36] SHI, Yuhui; EBERHART, Russell C. Empirical study of particle swarm optimization. In: *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*. IEEE, 1999.

[37] KENNEDY, James. The particle swarm: social adaptation of knowledge. In: *Evolutionary Computation, 1997., IEEE International Conference on*. IEEE, 1997. p. 303-308.

[38] SHI, Yang, et al. Cellular particle swarm optimization. *Information Sciences*, 2011, 181.20: 4460-4493.

[39] LIANG, Jing J., et al. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *Evolutionary Computation, IEEE Transactions on*, 2006, 10.3: 281-295.

[40] PERAM, Thanmaya; VEERAMACHANENI, Kalyan; MOHAN, Chilukuri K. Fitness-distance-ratio based particle swarm optimization. In: *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*. IEEE, 2003. p. 174-181.

[41] VAN LAARHOVEN, Peter JM; AARTS, Emile HL. *Simulated annealing*. Springer Netherlands, 1987.

[42] MASSA, A.; PASTORINO, M.; RANDAZZO, A. Reconstruction of two-dimensional buried objects by a differential evolution method. *Inverse Problems*, 2004, 20.6: S135.

[43] STORN, Rainer. Differential evolution design of an IIR-filter. In: *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*. IEEE, 1996. p. 268-273.

[44] LI, X.; YIN, M. Optimal synthesis of linear antenna array with composite differential evolution algorithm. *Scientia Iranica*, 2012, 19.6: 1780-1787.

[45] SOTIROUDIS, Sotirios P., et al. Application of a composite differential evolution algorithm in optimal neural network design for propagation path-loss prediction in mobile communication systems. *Antennas and Wireless Propagation Letters, IEEE*, 2013, 12: 364-367.

[46] GOUDOS, Sotirios K., et al. Self-adaptive differential evolution applied to real-valued antenna and microwave design problems. *Antennas and Propagation, IEEE Transactions on*, 2011, 59.4: 1286-1298.

[47] KRINK, T.; VESTERSTROM, J. S.; RIGET, J. Particle swarm optimisation with spatial particle extension. In: *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*. IEEE, 2002. p. 1474-1479.

[48] WANG, Lin, et al. An Effective Hybrid Self-Adapting Differential Evolution Algorithm for the Joint Replenishment and Location-Inventory Problem in a Three-Level Supply Chain. 2013.

[49] DA SILVA, Adenilton J.; MINEU, Nicole L.; LUDERMIR, Teresa B. Evolving artificial neural networks using adaptive differential evolution. In: *Advances in Artificial Intelligence–IBERAMIA 2010*. Springer Berlin Heidelberg, 2010. p. 396-405.

[50] WEBER, Tiago Oliveira; VAN NOIJE, Wilhelmus AM. Design of Analog Integrated Circuits Using Simulated Annealing/Quenching with Crossovers and Particle Swarm Optimization. *Simulated Annealing-Advances, Applications and Hybridizations*, 2012.